

UNIVERSIDADE FEDERAL DA PARAÍBA
CENTRO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA

JOÃO MARTINS DE OLIVEIRA NETO

**VIDEOLIB: UM MIDDLEWARE ESCALÁVEL PARA O
DESENVOLVIMENTO DE APLICAÇÕES MULTIMÍDIA DE TEMPO
REAL**

João Pessoa

2018

Universidade Federal da Paraíba
Centro de Informática
Programa de Pós-Graduação em Informática

VideoLib: Um middleware escalável para o desenvolvimento
de aplicações multimídia de tempo real

João Martins de Oliveira Neto

Dissertação submetida à Coordenação do Curso de Pós-Graduação
em Informática da Universidade Federal da Paraíba como parte dos
requisitos necessários para obtenção do grau de Mestre em Informática.

Área de Concentração: Ciência da Computação
Linha de Pesquisa: Computação Distribuída

Lincoln David Nery e Silva
Tiago Maritan Ugulino de Araújo

João Pessoa, Paraíba, Brasil
©João Martins de Oliveira Neto, 30 de agosto de 2018

Catálogo na publicação
Seção de Catalogação e Classificação

O48v Oliveira Neto, João Martins de.
VideoLib: Um middleware escalável para o
desenvolvimento de aplicações multimídia de tempo real
/ João Martins de Oliveira Neto. - João Pessoa, 2019.
70 f.

Orientação: Tiago Maritan Ugulino de Araújo.
Coorientação: Lincoln David Nery e Silva.
Dissertação (Mestrado) - UFPB/CI.

1. Multimídia. 2. Videocolaboração. 3. Middleware. I.
de Araújo, Tiago Maritan Ugulino. II. Título.

UFPB/BC



UNIVERSIDADE FEDERAL DA PARAÍBA
CENTRO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA

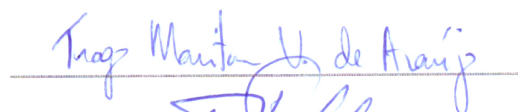


Ata da Sessão Pública de Defesa de Dissertação de Mestrado de João Martins de Oliveira Neto, candidato ao título de Mestre em Informática na Área de Sistemas de Computação, realizada em 30 de agosto de 2018.

1 Aos trinta dias do mês de agosto, do ano de dois mil e dezoito, às dezesseis horas, no
2 Centro de Informática da Universidade Federal da Paraíba, em Mangabeira, reuniram-se os
3 membros da Banca Examinadora constituída para julgar o Trabalho Final do Sr. João Martins
4 de Oliveira Neto, vinculado a esta Universidade sob a matrícula nº 2016100083, candidato
5 ao grau de Mestre em Informática, na área de "Sistemas de Computação", na linha de
6 pesquisa "Computação Distribuída", do Programa de Pós-Graduação em Informática, da
7 Universidade Federal da Paraíba. A comissão examinadora foi composta pelos professores:
8 Tiago Maritan Ugulino de Araujo (PPGI-UFPB), Orientador e Presidente da Banca, Rostand
9 Edson Oliveira Costa (PPGI-UFPB), Examinador Interno, Lincoln David Nery e Silva (UFPB),
10 Examinador Externo ao Programa, Daniel Faustino L De Souza (UFERSA), Examinador
11 Externo à Instituição. Dando início aos trabalhos, o Presidente da Banca, cumprimentou os
12 presentes, comunicou aos mesmos a finalidade da reunião e passou a palavra ao candidato
13 para que o mesmo fizesse a exposição oral do trabalho de dissertação intitulado "VideoLib:
14 Um middleware escalável para o desenvolvimento de aplicações multimídia de tempo real".
15 Concluída a exposição, o candidato foi arguido pela Banca Examinadora que emitiu o
16 seguinte parecer: "**aprovado**". Do ocorrido, eu, Claurton de Albuquerque Siebra,
17 Coordenador do Programa de Pós-Graduação em Informática, lavrei a presente ata que vai
18 assinada por mim e pelos membros da banca examinadora. João Pessoa, 30 de agosto de
19 2018.


Prof. Dr. Claurton de Albuquerque Siebra

Prof. Dr. Tiago Maritan Ugulino de Araujo
Orientador (PPGI-UFPB)



Prof. Dr. Rostand Edson Oliveira Costa
Examinador Interno (PPGI-UFPB)



Prof. Dr. Lincoln David Nery e Silva
Examinador Externo ao Programa (UFPB)



Prof. Dr. Daniel Faustino L De Souza
Examinador Externo à Instituição (UFERSA)



Resumo

Nos últimos anos, tem-se observado um aumento na oferta e demanda de aplicações baseadas na disseminação de vídeo pela internet. Com a popularização dessas tecnologias, outras áreas do conhecimento começaram a utilizar suas funcionalidades, surgindo assim ambientes colaborativos que utilizam o vídeo como meio principal na troca de informação.

De videoconferências até transmissões educacionais voltadas para a telemedicina, as etapas de configuração e implantação dessas aplicações costumam tomar muito tempo, além de comumente limitarem o número de participantes em uma mesma sessão.

Este trabalho propõe uma arquitetura escalável e um middleware que a utiliza que permitem o rápido desenvolvimento e implantação de aplicações de compartilhamento de vídeo em tempo real. Ele possui etapas de implantação mais simples que soluções tradicionais e suporta um número grande de usuários produzindo e consumindo vídeos simultaneamente.

Palavras-chave: Multimídia, Videocolaboração, Middleware.

Abstract

In the past few year there has been an increase in the demand for applications based on the distribution of video over the Internet. As these technologies became more popular, several areas of knowledge started to take advantage of their features. Thus, several collaborative environments have been developed using video streams as the media of communication.

From videoconferencing to educational telemedicine video transmissions, these collaborative applications usually take a long time to be configured and deployed. They also frequently limit the number of participants in a single session.

This dissertation proposes a scalable architecture and a middleware that enable the quick development and deployment of video sharing applications in real time. It provides a very low configuration overhead when compared to traditional solutions and supports a great number of users providing and consuming videos simultaneously.

Keywords: Multimedia, Video Collaboration, Middleware.

Agradecimentos

A minha família, por todo o suporte me dado durante toda a minha vida.

A meu orientador, pelas oportunidades, dicas e ensinamentos ao longo do desenvolvimento deste trabalho e de tantos outros.

A todos os professores que de alguma forma fizeram parte da minha formação acadêmica durante os meus anos de universidade.

A meus amigos e colegas, pelo companheirismo

Conteúdo

1	Introdução	10
1.1	Objetivos	12
1.1.1	Objetivo Geral	12
1.1.2	Objetivos Específicos	12
1.2	Estrutura do Trabalho	13
2	Fundamentação Teórica	14
2.1	O Modelo <i>Publish/Subscribe</i>	14
2.2	DDS	16
2.3	ContextNet	17
2.4	FFMPEG	19
2.5	GStreamer	20
3	Revisão bibliográfica	21
3.1	Transmissão de Vídeo Sobre o Publish/Subscribe	21
3.2	Soluções de Transmissão de Vídeo em Tempo Real	23
4	Modelos e Métodos	28
4.1	Descrição da solução proposta	28
4.2	Metodologia de Desenvolvimento	33
5	Implementação e Testes	35
5.1	Adaptações do ContextNet	35
5.2	Desenvolvimento da VideoLib	36
5.3	Testes Funcionais	38

5.4	Testes de Desempenho	40
5.5	Mega-conferência	42
6	Resultados	45
6.1	Funcionalidades	45
6.2	Desempenho	46
6.3	Análise Comparativa	48
6.4	Mega-conferência	50
7	Criando Uma Aplicação com a VideoLib	53
7.1	A aplicação	53
7.2	Desenvolvimento	54
7.3	Cenário de Uso	55
8	Conclusão	58
	Bibliografia	64
A	API da VideoLib	65
B	Arquivo de configuração do Gerenciador de transformações	67
C	Código para a criação de uma Transformação	69

Lista de Símbolos

API : Interface de Programação de Aplicações

CNClib**** : *ClientLib*

DDS : Serviço de Distribuição de Dados

DLNA : *Digital Living Network Alliance*

GD : *Group Definer*

GW : *Gateway*

MN : *Mobile Node*

MR-UDP : *Mobile Reliable User Datagram Protocol*

OMG : *Object Management Group*

PM : *PoA Manager*

PoA : Ponto de Acesso

Pub/Sub : *Publish/Subscribe*

QoS : Qualidade de Serviço

RNP : Rede Nacional de Ensino e Pesquisa

RUDP : *Reliable User Datagram Protocol*

SDDL : *Scalable Data Distribution Layer*

TCP : *Transmission Control Protocol*

UDI : *Universal DDS Interface*

UDP : *User Datagram Protocol*

Lista de Figuras

2.1	Esquema básico do modelo <i>Pub/Sub</i>	15
2.2	Arquitetura do SDDL [Silva 2014]	17
2.3	Componentes do SDDL [David et al. 2013]	19
3.1	Resultados obtidos por Detti [2][1][Detti et al. 2010]	22
3.2	Performance Telemática	24
3.3	Versus: Espetáculo colaborativo de dança [RNP 2005]	25
3.4	Interface web do Arthron [Filho et al. 2012]	26
3.5	Transmissão de telemedicina via Arthron	27
4.1	Componentes e dependências da VideoLib.	29
4.2	Visão geral do funcionamento da VideoLib.	30
4.3	Esquema de proxies da VideoLib.	32
4.4	Gerenciadores de Transformações.	33
5.1	Componentes do Gstreamer que formam aplicação do mosaico	40
5.2	Geração aninhada de mosaicos	41
5.3	Diagrama de Classe dos componentes da VideoLib.	43
5.4	Diagrama de classes do controlador da VideoLib.	44
6.1	Transformação de <i>chroma key</i> ao vivo	46
6.2	Transformação de mosaico ao vivo	47
6.3	Variação da latência com a taxa de saída	49
6.4	Performance telemática utilizando a VideoLib.	50
6.5	Modelagem do cenário do caso de uso	51
6.6	Utilização de recursos das máquinas da mega-conferência	52

LISTA DE FIGURAS

6.7	Fluxo final do cenário de uso da mega conferência	52
7.1	Visão geral da aplicação.	54
7.2	Processo de criação de Transformações.	57

Lista de Tabelas

5.1	Tópicos de transmissão de dados do <i>middleware</i>	36
6.1	Latência e taxa de quadros perdidos para transmissões pela <i>VideoLib</i>	48

Lista de Códigos Fonte

5.1	Exemplo de código do controlador	38
7.1	Criando componentes da <i>VideoLib</i>	55
7.2	Configurando as listas de componentes	55
7.3	Configurando botões da aplicação	56
B.1	Exemplo de arquivo de configuração do gerenciador de transformações . . .	67
C.1	Criação e inicialização de uma transformação	69

Capítulo 1

Introdução

Aplicações baseadas na transmissão de vídeo em rede vêm ocupando cada vez mais espaço na internet nos últimos anos. De acordo com projeções apresentadas em um estudo feito pela CISCO, em 2020 a porcentagem do tráfego de dados pela Internet destinada à transmissão de vídeo será de 82%. Isto equivale a cerca de 1 milhão de minutos de vídeo trafegando na rede a cada segundo [Cisco 2015]. Estas transmissões podem consistir tanto de vídeo sob demanda, como são oferecidos pelo YouTube¹ e pelo Netflix², como de transmissões de vídeo ao vivo como fazem o *Twitch.TV*³ e o *Mixer*⁴.

O amadurecimento das tecnologias de transmissão de vídeo fez com que elas começassem a ser utilizadas em novos cenários, possibilitando o surgimento de novas funcionalidades em aplicações dos mais diversos tipos. Uma das áreas que se beneficia dessas tecnologias é a telemedicina. Ela consiste na utilização de sistemas de telecomunicações e da tecnologia da informação para a promoção de atividades assistenciais e educacionais para profissionais da saúde e seus pacientes, diminuindo distâncias e possibilitando a oferta de serviços em áreas remotas que não teriam acesso aos mesmos de outra maneira [Wen 2008]. Em 2012, por exemplo, cerca de metade dos hospitais dos Estados Unidos possuíam programas de telemedicina [Adler-Milstein, Kvedar e Bates 2014].

Ainda tratando-se de ambientes médicos, um projeto piloto realizado em Queensland, nos Estados Unidos, mostrou que a videoconferência pode ser adotada para facilitar

¹<http://www.youtube.com>

²<http://www.netflix.com>

³<http://twitch.tv>

⁴<http://mixer.com>

o supervisionamento de operadores de máquinas de raio X. Em seus experimentos, especialistas ofereceram suporte remoto aos operadores em questões como o posicionamento dos pacientes e a seleção do nível de exposição à radiação, melhorando a qualidade das imagens resultantes e diminuindo a quantidade de pessoas expostas à radiação dessas máquinas. [Rawle et al. 2017]. Também foram encontrados estudos que sugerem que a utilização de sistemas de videoconferência no treinamento de pais de crianças com o “Transtorno do Deficit de Atenção com Hiperatividade” pode ser tão eficaz quanto sessões presenciais de mesmo conteúdo, podendo se tornar uma ferramenta essencial para a educação em regiões remotas e com maior dificuldade no acesso à informação [Xie et al. 2013].

Um outro tipo de aplicação que utiliza estes tipos de tecnologias são as voltadas para performances artísticas telemáticas. Os relatórios de Ubik [Ubik et al. 2016] mostram que músicos e dançarinos podem colaborar remotamente em um espetáculo artístico, desde que sejam observadas algumas condições como o retardo da transmissão dos dados. Por fim, sistemas de videoconferência voltados para usuários domésticos e para empresas são exemplos de aplicações colaborativas baseadas em vídeo que facilitam a vida e fazem parte do cotidiano de diversas pessoas, diminuindo distâncias e custos de comunicação.

O desenvolvimento desses tipos de aplicações faz com que novos cenários mais complexos e dinâmicos comecem a surgir, trazendo consigo novos requisitos que podem não ser atendidos de forma satisfatória pelas soluções existentes. Um desses requisitos é a compatibilidade entre diferentes tipos de dispositivos. Usuários comuns possuem cada vez mais telas e computadores disponíveis, e nem sempre as aplicações são compatíveis com todos os seus dispositivos. Outro requisito é o suporte a um número elevado de usuários. O *Skype*, uma das soluções de videoconferência mais populares, possui um limite de 25 usuários simultâneos em uma chamada de vídeo⁵, por exemplo. Esta variedade e dinamicidade dos cenários em que estas soluções são empregadas tornam o desenvolvimento de novas aplicações de transmissão de conteúdo multimídia uma tarefa complicada para desenvolvedores de *software*.

⁵<https://www.skype.com/pt-br/features/group-calls/>

1.1 Objetivos

Dadas as limitações previamente apresentadas, este trabalho propõe uma arquitetura e um *middleware* que suporta o desenvolvimento e a implantação de aplicações multimídia escaláveis baseadas na disseminação de vídeo em tempo real. A arquitetura proposta é uma extensão do *ContextNet*, um *middleware* que permite a implantação de aplicações escaláveis originalmente desenvolvido para a disseminação de dados estruturados com foco em informações de contexto [David et al. 2012]. Dessa forma, este trabalho serve como uma prova de conceito da utilização do ContextNet para a transmissão de vídeo em larga escala. Uma vez que esta estrutura esteja implementada, diversas aplicações podem fazer uso da mesma, como será mostrado ao longo deste trabalho.

Para simplificar o desenvolvimento de aplicações utilizando o *middleware* desenvolvido é proposta também uma interface de programação de aplicações (API) que gerencia as fases de captura, disseminação, processamento e exibição de vídeo em tempo real, comumente presentes nas aplicações dessa natureza, de maneira similar ao que o *ContextNet* faz para informações de contexto. Também faz parte do trabalho o desenvolvimento de uma aplicação baseada neste *middleware*.

1.1.1 Objetivo Geral

Prover um *middleware* para aplicações multimídia distribuídas e escaláveis baseado no ContextNet.

1.1.2 Objetivos Específicos

- Adaptar o *ContextNet* para a distribuição de vídeo em tempo real
- Desenvolver uma camada do *middleware* para o gerenciamento de sessões de colaboração entre diversos participantes, com suporte a captura, transmissão, processamento e exibição de fluxos de vídeo.
- Propor uma API para Facilitar o desenvolvimento de aplicações que utilizam esse sistema.
- Desenvolver uma aplicação modelo baseada nesta API.

1.2 Estrutura do Trabalho

O restante do trabalho está organizado da seguinte maneira:

- No Capítulo 2 serão apresentados conceitos e ferramentas utilizados ao longo do trabalho na implementação da solução proposta.
- No Capítulo 3 será feita uma revisão da literatura para elencar algumas soluções de distribuição de vídeo e analisar estudos sobre a transmissão de vídeo relevantes para este trabalho.
- No Capítulo 4 estará descrita a solução proposta neste trabalho e as metodologias de avaliação que serão utilizadas.
- No Capítulo 5 serão descritos alguns detalhes da implementação da solução proposta no capítulo anterior
- No capítulo 6 serão apresentados e avaliados os resultados do trabalho obtidos pelos testes descritos anteriormente.
- O Capítulo 7 é destinado a exemplificar o processo de criação de uma aplicação que utiliza a solução proposta e desenvolvida neste trabalho
- No Capítulo 8, por fim, serão feitas as considerações finais, enunciando alguns trabalhos para o futuro.

Capítulo 2

Fundamentação Teórica

Neste capítulo são explicados os conceitos e tecnologias utilizados no decorrer deste trabalho. Dentre eles tem-se as tecnologias voltadas para a distribuição de dados em larga escala e as ferramentas para o desenvolvimento de aplicações multimídia.

2.1 O Modelo *Publish/Subscribe*

Um sistema *Publish/Subscribe* (*Pub/Sub*) consiste num conjunto de nós que trocam informações entre si em um ambiente distribuído e um serviço de entrega de mensagens, que serve para promover o desacoplamento desses nós. Os clientes desse tipo de sistema podem atuar como produtores (*publishers*), que geram conteúdo, e consumidores (*subscribers*), que recebem conteúdo do sistema [Banavar et al. 1999]. Um mesmo cliente pode assumir ambos os papéis durante sua execução.

Os produtores enviam mensagens através da ação denominada *publish*, ou publicação. Os consumidores, por sua vez, realizam o *subscribe*, ou inscrição, para sinalizar ao serviço qual o tipo de mensagem em que possuem interesse, evitando que sejam recebidas informações indesejadas. Nesse modelo de comunicação os clientes se tornam independentes entre si. Produtores, ao enviar uma mensagem para o serviço, não tem conhecimento de quais ou quantos consumidores irão recebê-la. Analogamente, os consumidores recebem as mensagens que lhe dizem respeito sem necessariamente saber quais produtores as enviaram [Eugster et al. 2003].

A interação entre os clientes é exemplificada na Figura 2.1. Nela pode-se observar dois

produtores enviando mensagens marcadas como "clima" e um terceiro produtor enviando uma mensagem marcada como "localização" para um serviço de entrega de mensagens. Do outro lado, os consumidores avisam ao serviço quais as mensagens que os interessam. Um deles deseja receber mensagens de clima e o outro mensagens de localização. O serviço, por sua vez, redireciona as mensagens dos produtores para os consumidores correspondentes.

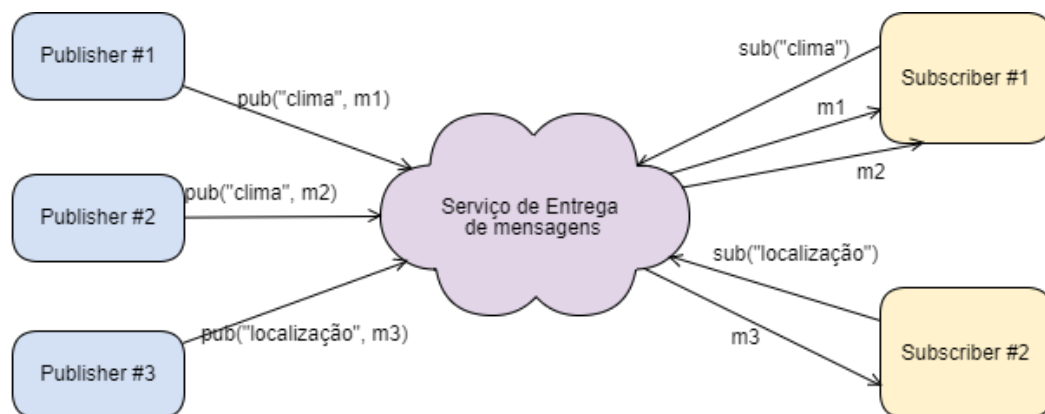


Figura 2.1: Esquema básico do modelo *Pub/Sub*

A independência que este modelo provê para os seus usuários pode ser justificada pelo desacoplamento que ele oferece em três níveis:

- **Tempo:** Produtores e consumidores não precisam estar conectados ao sistema simultaneamente;
- **Espaço:** Ambas as partes não precisam conhecer a localização (endereço IP, por exemplo) das outras.
- **Sincronia:** Produtores e consumidores trocam mensagens de forma assíncrona, sem a necessidade de que esperem pelo envio ou recebimento das mesmas

A inscrição de um consumidor em uma determinada categoria de mensagens pode se dar de várias formas. As mais comumente utilizadas são as baseadas em *tópicos*, *conteúdos* e *tipos*. No primeiro caso, o mais utilizado, cada mensagem possui um tópico, que pode ser uma sequência arbitrária de caracteres ou uma espécie de domínio pré-definido. Os consumidores utilizam esses tópicos para identificar as mensagens que desejam receber. Um consumidor pode, por exemplo, requisitar um tópico "temperatura" para receber informações climáticas de sensores diversos. Também é possível utilizar expressões regulares para

abranjer vários tópicos em uma única inscrição [Huang e Garcia-Molina 2004]. Já nas requisições baseadas em conteúdo os consumidores definem regras de acordo com propriedades das mensagens em si. Por exemplo, se a mensagem possuir um campo "temperatura", um consumidor pode definir que deseja receber mensagens onde este campo é maior que um valor determinado. No último caso, onde as inscrições são baseadas em tipos, cada classe de mensagem possui uma estrutura pré-definida (podendo haver subclasses de mensagens), e os consumidores apenas recebem as mensagens das classes nas quais se inscreveram [Eugster et al. 2003].

Devido à essa versatilidade e adaptabilidade a ambientes dinâmicos, o modelo *Pub/Sub* se apresenta como uma alternativa escalável para a disseminação de dados em ambientes móveis, sendo bastante utilizado e há bastante tempo [Huang e Garcia-Molina 2004].

2.2 DDS

Uma das soluções que se baseia no modelo *Pub/Sub* de comunicação é o padrão de *middleware Data Distribution Service for Real-Time Systems* (DDS) ¹, desenvolvido pela *Object Management Group* (OMG). Ele conecta os componentes de um sistema distribuído, provendo uma conectividade de baixa latência, alta confiabilidade e escalabilidade. Isso faz com que ele seja bastante utilizado na Internet das Coisas (IoT) [Pardo-Castellote 2003].

Pertencente à categoria dos *middlewares* orientados a dados, o DDS implementa as funcionalidades do *Pub/Sub* discutidas anteriormente, permitindo que produtores identifiquem os objetos que estão enviando e que consumidores especifiquem quais objetos desejam receber. Ele também possibilita a criação de tópicos e o gerenciamento de políticas de qualidade de serviço (QoS) [OMG 2006].

Outra facilidade que o DDS fornece a seus usuários é a descoberta dinâmica. Isso significa que a aplicação encontra automaticamente os seus clientes, sem precisar conhecê-los ou configurá-los previamente. Esta descoberta pode acontecer em tempo de execução, caracterizando o que se chama de "plug and play". Para isso os clientes precisam estar na mesma máquina ou conectados em uma mesma rede. Algumas das implementações

¹<http://www.omg.org/omg-dds-portal/>

mais adotadas do DDS são o *OpenSplice*² e o RTI³ [Dworak et al. 2011]

2.3 ContextNet

Este trabalho foi desenvolvido tendo como base o projeto *ContextNet*, desenvolvido na PUC-Rio. Ele teve como objetivo o desenvolvimento de serviços baseados em contexto para aplicações colaborativas de larga escala, como por exemplo o monitoramento e coordenação de entidades móveis, que podem ser desde usuários de dispositivos portáteis (*smartphones*) até veículos ou robôs autônomos. Para isso, o projeto focou no desenvolvimento de um *middleware* que permite a distribuição e a categorização de informações de contexto de forma escalável, dando suporte a centenas de milhares de entidades [Endler et al. 2011].

A parte do *ContextNet* explorada neste trabalho é a sua camada de comunicação, o *Scalable Data Distribution Layer* (SDDL) [David et al. 2012], ilustrado na Figura 2.2. Ele consiste basicamente em um *middleware* que conecta nós estacionários de uma rede DDS cabeada a nós móveis com conexões IP sem fio. Esta conexão sem fio é realizada através do protocolo *Mobile Reliable UDP* (MR-UDP).

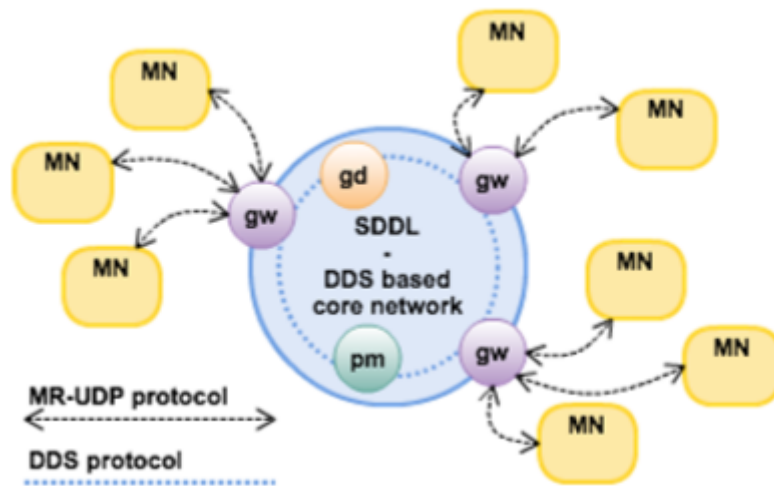


Figura 2.2: Arquitetura do SDDL [Silva 2014]

O MR-UDP é um protocolo que provê uma comunicação confiável entre nós móveis

²<http://www.prismtech.com/vortex/vortex-opensplice>

³<https://www.rti.com/products/dds>

baseada no UDP com sobrecarga mínima. Ele incrementa outro protocolo, o *Reliable UDP* (RUDP) [Bova e Krivoruchka 1999], com funcionalidades voltadas para ambientes móveis como a capacidade de lidar com conectividade intermitente, a travessia de NAT e o uso reduzido de recursos de dispositivos móveis. Ele não foi projetado, entretanto, para a transmissão contínua de dados, como nos casos de fluxos multimídia [Silva, Endler e Roriz 2013].

Ainda na Figura 2.2 podem-se observar três tipos de nós na rede estacionária baseada no DDS. Estes são:

- *Gateway (GW)*: São os pontos de acesso (PoA) dos nós móveis à rede principal. Cada nó móvel que deseja se conectar ao SDDL deve estabelecer uma conexão MR-UDP com um *gateway*. Este então irá traduzir as mensagens entre os protocolos DDS e MR-UDP, estabelecendo assim a integração da rede principal com os dispositivos móveis. Também cabe aos GWs avisar aos outros membros da rede principal quando um novo nó móvel se conecta ou desconecta do sistema.
- *PoA-Manager (PM)*: Periodicamente distribuem uma lista de GWs disponíveis para os nós móveis, em ordem de preferência de conexão. Dessa forma, conseguem anunciar quando um novo PoA é criado ou destruído. Também conseguem, ao enviar listas diferentes para cada nó, balancear a carga da rede entre os PoAs.
- *GroupDefiner (GD)*: Avaliam a participação de cada nó em grupos criados dinamicamente. Cada aplicação define as regras para a criação dos grupos. Uma aplicação pode criar grupos, por exemplo, baseados na localização geográfica dos nós, agrupando clientes de uma mesma cidade, estado ou país. Essa lista de membros é então distribuída para todos os *gateways*, permitindo o envio de mensagens específicas para grupos específicos (*groupcast*).

Do lado dos nós móveis (MN), uma aplicação cliente utiliza a ClientLib (CNCLib), uma biblioteca para estabelecer e gerenciar conexões MR-UDP e trocar informações com os *gateways*. Já do lado da rede estacionária, todos os componentes utilizam a *Universal DDS Interface* para se comunicar através das diversas soluções existentes que implementam o DDS. Todos os componentes citados nesta seção podem ser visualizados na imagem 2.3 [David et al. 2013].

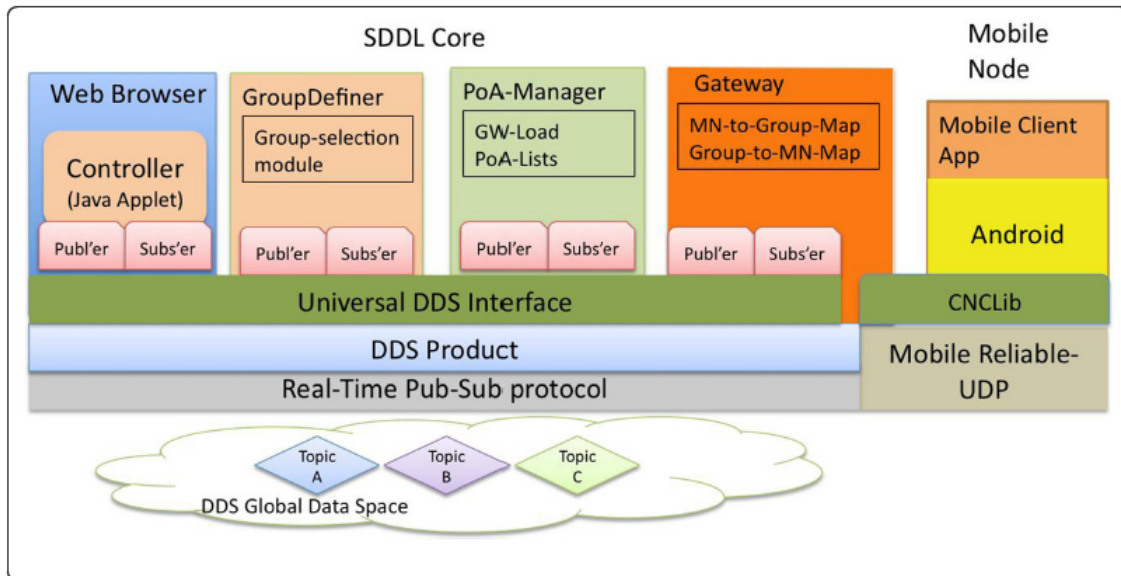


Figura 2.3: Componentes do SDDL [David et al. 2013]

2.4 FFMPEG

Nos últimos anos o FFMPEG⁴ tem se tornado uma das ferramentas mais utilizadas no desenvolvimento de aplicações multimídia. Alguns exemplos podem ser facilmente encontrados na literatura em aplicações de aquisição [Xu e Cao 2014], processamento [Hock e Lingxia 2014], transmissão [Zhao, Zhou e Jin 2015] e análise [Xiaohua, Xiuhua e Caihong 2013] de vídeos. A popularidade do FFMPEG pode ser explicada pelo fato de que ele suporta praticamente todos os formatos mais populares tanto de codificadores quanto de encapsuladores de vídeo. Ele pode ser utilizado tanto pela linha de comando do sistema operacional como através de suas APIs que possibilitam o desenvolvimento de aplicações multimídia.

No decorrer deste trabalho o FFMPEG foi utilizado para desenvolver algumas das aplicações que dão suporte à solução proposta, como por exemplo uma aplicação que cria um fluxo de vídeo a partir de um arquivo ou de uma *webcam* e a fornece para o sistema desenvolvido.

⁴<http://www.ffmpeg.org/>

2.5 GStreamer

Outra ferramenta bastante utilizada no desenvolvimento de aplicações multimídia é o *GStreamer*⁵[Otnes, Eastwood e Colin 2015] [Nimmi et al. 2014] [Wang, Zhang e Ge 2014]. Ele provê uma forma fácil de criar aplicações multimídia através do encadeamento de seus componentes, denominados filtros. Para reproduzir um fluxo de vídeo, por exemplo, liga-se um filtro que recebe o fluxo de um arquivo ou da rede à outro que efetue a decodificação do fluxo. Posteriormente, liga-se esse último a um filtro que exibe o conteúdo processado.

Ele foi utilizado nesse trabalho para a criação de transformadores de vídeo, que recebem um ou mais fluxos multimídia e geram um fluxo resultante através da combinação e transformação das entradas.

⁵<http://gstreamer.freedesktop.org/>

Capítulo 3

Revisão bibliográfica

Neste capítulo será apresentada uma revisão bibliográfica feita sobre o tema deste trabalho. Essa revisão foi realizada de maneira *ad-hoc* através de pesquisas no portal de periódicos da CAPES ¹, no Google Acadêmico ² e nas bibliotecas digitais ACM.DL³ e IEEE Xplore⁴. Também foram encontrados trabalhos a partir das referências e citações dos resultados das pesquisas.

Primeiramente é feito um levantamento de trabalhos que exploram a transmissão de vídeo sobre o modelo Pub/Sub (termos-chave: "*real-time*", "*video transmission*", "*publish subscribe*", "*DDS*", "*scalable*") e posteriormente são enumerados alguns trabalhos que utilizam soluções de transmissão de vídeo em tempo real, exemplificando dessa forma casos de uso reais para a solução proposta (termos-chave: "*cyber performances*", "*telemedicine*", "*video transmission*", "*video conferencing*", "*video collaboration*", "*multimedia streams*").

3.1 Transmissão de Vídeo Sobre o Publish/Subscribe

Em seus trabalhos, Al-Madani [Al-Madani, Al-Saeedi e Al-Roubaiey 2013] [Al-Madani, Al-Roubaiey e Baig 2014] analisa a transmissão de vídeo em tempo real no modelo *Publish Subscribe* sobre o DDS. Ele utiliza métricas de avaliação como utilização de banda e jitter para comparar a transmissão através do DDS com a transmissão feita pelo *software* VLC.

¹<http://www.periodicos.capes.gov.br/>

²<https://scholar.google.com.br/>

³<https://dl.acm.org/>

⁴<http://ieeexplore.ieee.org>

É chegada à conclusão que o DDS é um forte candidato a ser usado para este propósito. O trabalho, entretanto, testa apenas um número pequeno de *subscribers* (15) devido a limitações técnicas.



Figura 3.1: Resultados obtidos por Detti [Detti et al. 2010]

Detti [Detti et al. 2010], por sua vez, estuda o envio de vídeo codificado pelo *H264 Scalable Video Coding* sobre o DDS em redes sem fio. Em seus experimentos ele compara a qualidade da transmissão de vídeo H.264 sobre o DDS com a transmissão de vídeo MPEG-2. Seus resultados mostram que com o aumento o aumento do fluxo de dados e consequente sobrecarga da rede, as transmissões MPEG-2 perdem qualidade muito mais rápido que as transmissões através do DDS devido aos mecanismos de qualidade de serviço oferecidos pelo mesmo, como mostrado na Figura 3.1.

Garcia [García-Valls, Basanta-Val e Estévez-Ayres 2010] propõe uma arquitetura de *streaming* adaptativo de vídeo sobre o DDS. Sua solução é voltada para a disseminação de vídeos de câmeras de segurança. Ele exalta as garantias de qualidade de serviço ofertadas pelo DDS quando comparado a outros sistemas.

Esses trabalhos comprovam que a utilização do modelo *Publish/Subscribe* e do DDS é, de fato, viável para a transmissão de vídeo. Os resultados obtidos pelos autores estudados são sempre positivos quando comparados à transmissão tradicional utilizando protocolos como UDP e RTP.

3.2 Soluções de Transmissão de Vídeo em Tempo Real

Alguns relatos de experiências e aplicações baseadas na transmissão de vídeo em tempo real podem ser encontradas no meio acadêmico. Um exemplo disso é o trabalho de Ubik [Ubik et al. 2016], que utiliza soluções de transmissão de vídeo para avaliar a viabilidade da execução de performances artísticas colaborativas em rede. Ele analisa diversos experimentos envolvendo dança e música colaborativa e conclui que, observadas as latências, os artistas conseguem interagir de forma satisfatória. Ele utilizou em seus experimentos os softwares *Ultragrid* [Holub et al. 2012] e *4K Gateway* [Halak e Ubik 2009] para realizar a transmissão dos fluxos de vídeo. A partir da descrição de suas demonstrações, percebe-se que este tipo de evento requer muitas etapas de configuração do ambiente. Em uma demonstração de dança em 2014, envolvendo quatro cidades distintas, foram utilizados quatro projetores e dois dispositivos de captura apenas na cidade onde o evento principal estava acontecendo.

A Figura 3.2 mostra um cenário hipotético similar aos relatados por Ubik. Nele, têm-se três ambientes diferentes que se comunicam através de transmissões multimídia. Nos ambientes (a) e (c) dançarinos são filmados e suas imagens são enviadas para um telão no ambiente (b). Uma dançarina no ambiente (b) interage com as projeções de seus colegas, e o resultado é filmado e transmitido para os outros ambientes de forma que todos os dançarinos obtenham um *feedback* da apresentação.

O *Ultragrid*, utilizado nos experimentos relatados por Ubik, é um *software* de codificação, transmissão, recepção e reprodução de fluxos de áudio e vídeo. Desenvolvido na Universidade de Masaryk, na República Tcheca, ele efetua transmissões de alta qualidade com baixa latência e é utilizado em ambientes colaborativos, atividades educacionais e aplicações gerais de transmissão de vídeo [Gharai et al. 2006]. O *Ultragrid*, entretanto, não apresenta funcionalidades de gerenciamento de sessões de videocolaboração. Assim sendo, cada fluxo gerado pelo programa deve ser manualmente configurado nos pontos de envio e de reprodução. Em ambientes como os relatados anteriormente onde muitos fluxos precisam ser configurados, a implantação do *Ultragrid* sem um sistema de gerenciamento de fluxos requer um investimento de tempo para a preparação dos ambientes e configuração dos softwares.

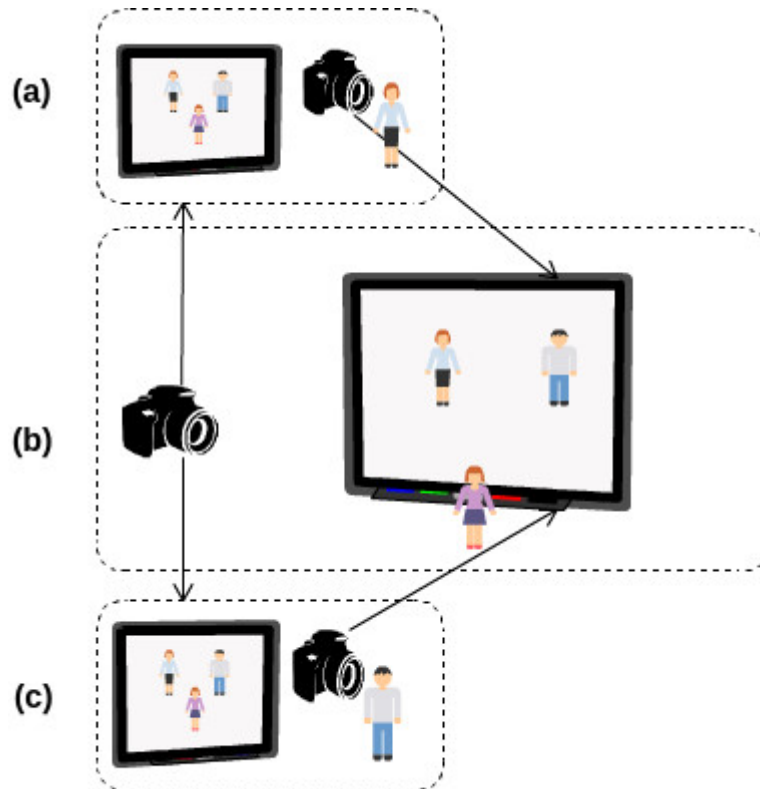


Figura 3.2: Performance Telemática

Além dos experimentos de Ubik, outros exemplos de performances artísticas colaborativas que fazem uso de fluxos de vídeo também são encontrados na literatura. Em 2005 foi realizado pela Rede Nacional de Ensino e Pesquisa (RNP), no evento de lançamento de sua rede multigigabit, o espetáculo *Versus*. Nele, bailarinas no local do espetáculo interagiam com outras em lugares remotos através de uma projeção de vídeo, como mostrado na Figura 3.3 [RNP 2005]. Outro exemplo de performance telemática envolvendo diversos participantes é o *Dancing Beyond Time*, realizado em 2013 e tendo, simultaneamente, participantes localizados no Brasil, Coreia, República Tcheca e Espanha [Santana 2014].

Um sistema que pode ser utilizado para auxiliar a colaboração entre usuários é o *Arthron*, uma ferramenta para a transmissão e gerenciamento remoto de fluxos de vídeo. Ele é composto por *encoders* e *decoders* que são gerenciados remotamente por um *manager*, capaz de controlar a transmissão e reprodução dos fluxos multimídia através de uma interface web. Essa interface possibilita a ligação entre as fontes de vídeo e as telas onde eles devem ser



Figura 3.3: Versus: Espetáculo colaborativo de dança [RNP 2005]

exibidos [Melo et al. 2010]. Essa interface web é mostrada na Figura 3.4. Ele já foi utilizado em alguns espetáculos colaborativos [Tavares 2015] e também em cenários da telemedicina [Filho et al. 2012]. Sua interface é simples e acessível a usuários que não possuem conhecimento aprofundado na área de informática e programação, como pode ser o caso de integrantes de performances artísticas e de médicos e estudantes de medicina. O *Arthron* não oferece, entretanto, uma arquitetura escalável, capaz de assegurar que um grande número de usuários consiga participar de uma sessão sem problemas na qualidade de serviço. Isso se deve ao fato de que a quantidade usual de participantes nos cenários para os quais ele foi idealizado não costuma ser muito grande. Além dessa limitação, cada componente do sistema deve ser previamente configurado e disponibilizado no ambiente web, adicionando mais uma etapa de configuração e dificultando a inserção em larga escala de novos participantes.

A Figura 3.5 ilustra um cenário, também hipotético, em que o *Arthron* é utilizado para gerenciar uma aplicação de telemedicina. Nesta aplicação, várias câmeras são espalhadas em uma sala de um centro cirúrgico e devem transmitir suas imagens para telas em dois auditórios diferentes. As setas laranjas da imagem mostram que cada uma das câmeras replica o seu fluxo multimídia para cada uma das telas. Além disso, as setas azuis retratam a comunicação do *manager* com cada um dos componentes envolvidos. Caso houvesse uma nova tela no cenário, ela deveria primeiramente se conectar à interface web para então o operador do sistema conectá-la a cada uma das câmeras. Este processo não é ideal para cenários onde o número de câmeras e tela cresce muito.

Além dos trabalhos acadêmicos encontrados, há também os sistemas comerciais de

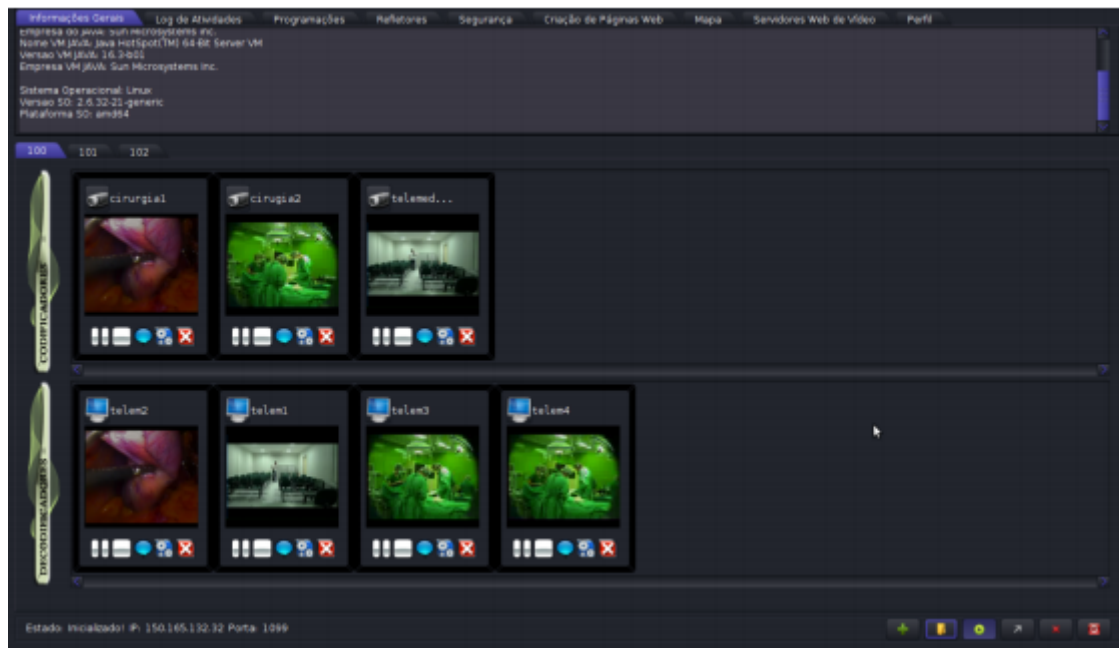


Figura 3.4: Interface web do Arthron [Filho et al. 2012]

videoconferência. Alguns desses sistemas são mais voltados para uso doméstico e não necessitam de nenhum equipamento extra, utilizam apenas a *webcam* de um computador ou a câmera de um *smartphone*, como o Skype ⁵ e o Hangouts ⁶. Por outro lado, também existem sistemas que são mais voltados para uso empresarial e geralmente utilizam salas e aparelhos projetados especificamente para videoconferências. Exemplo deste tipo de sistema é o Realpresence, da Polycom ⁷. Ambos os sistemas atendem bem os requisitos dos cenários para os quais foram projetados, mas não são suficientes para os cenários levantados nesta seção.

Um dos problemas destes sistemas é a escalabilidade. Eles costumam limitar o número de participantes em uma mesma sessão. O Skype e o Hangouts, por exemplo, permitem apenas 25 participantes de vídeo ativos simultaneamente em uma mesma sessão. Eles também não oferecem a flexibilidade necessária para se adaptar a qualquer aparelho que consiga capturar e reproduzir vídeo. Por fim, nenhum desses sistemas oferece uma API que possibilite a criação de novas aplicações independentes que atendam às necessidades específicas de um usuário diferenciado.

⁵<http://www.skype.com>

⁶<http://hangouts.google.com>

⁷<http://www.polycom.com>

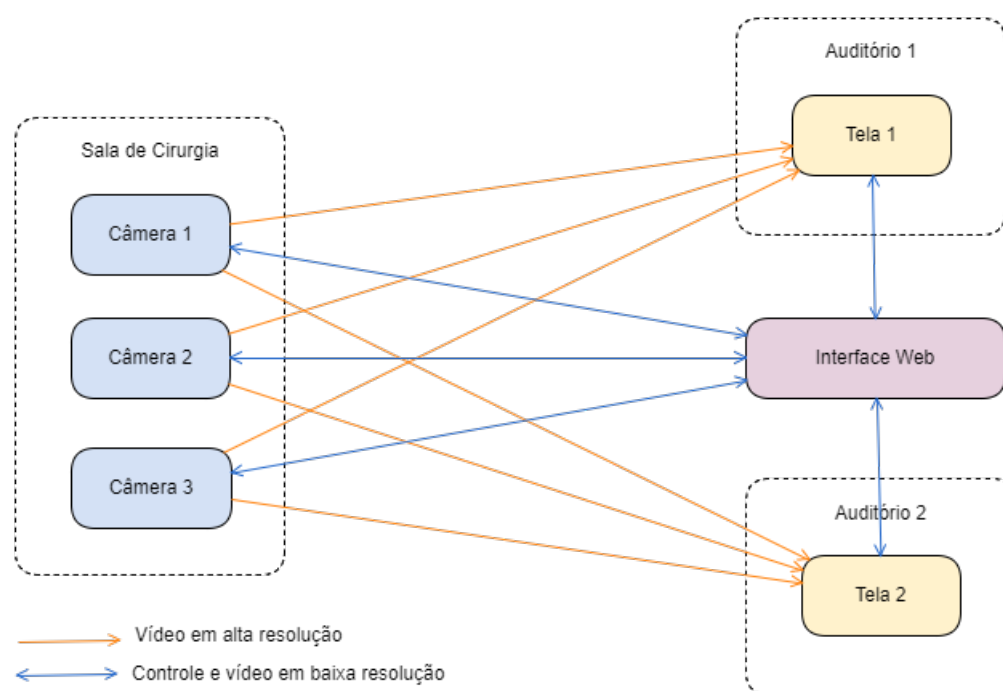


Figura 3.5: Transmissão de telemedicina via Arthron

Capítulo 4

Modelos e Métodos

Neste capítulo será descrita a solução de distribuição de vídeo proposta pelo trabalho e as etapas de seu desenvolvimento, desde sua modelagem até as metodologias utilizadas para a sua avaliação.

4.1 Descrição da solução proposta

Como solução para os cenários propostos no Capítulo 1, este trabalho propõe a VideoLib, uma extensão do *ContextNet* que possibilita, através de uma API, a construção de aplicações para o controle da captura, disseminação, processamento e exibição de vídeo em tempo real. Esta API foi pensada para ser suficientemente simples, tendo em vista que seus usuários nem sempre possuirão experiência ou conhecimento aprofundado das tecnologias que envolvem o desenvolvimento de aplicações multimídia.

Como mostrado na Figura 4.1, esse sistema utiliza o *ContextNet* como meio de comunicação entre os seus clientes devido à facilidade que ele provê na disseminação de conteúdo através do modelo *Pub/Sub*. Esse modelo possibilita que sejam criados canais de vídeo e de controle para cada aplicação, simplificando assim a comunicação entre os usuários do sistema. Por ser baseada no *Pub/Sub*, toda a comunicação entre os participantes é gerenciada pelo *middleware* e ocorre de forma desacoplada. Os clientes não precisam saber da localização nem da existência uns dos outros. Um nó produtor de conteúdo apenas sabe que os fluxos multimídia gerados por ele estarão disponíveis na rede, não importando quais outros nós o consumirão. Analogamente, um nó consumidor apenas precisa se inscrever

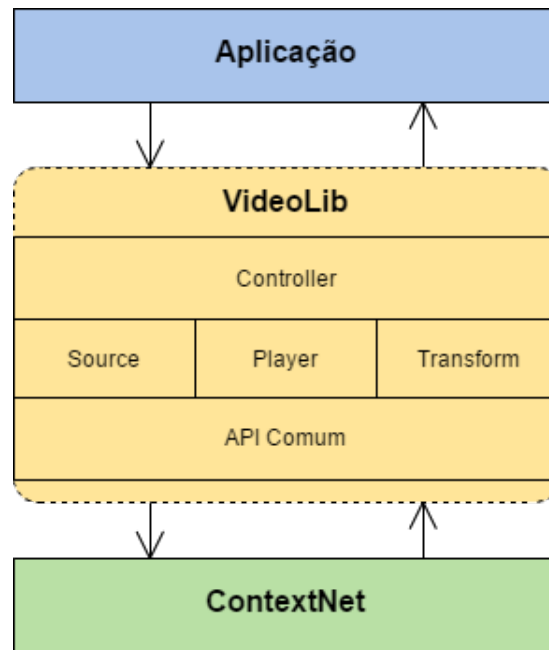


Figura 4.1: Componentes e dependências da VideoLib.

nos tópicos do conteúdo que tiver interesse, sem precisar de conhecimento prévio sobre os produtores. Essa propriedade da solução diminui o trabalho no que diz respeito ao tempo utilizado nas etapas de configuração das aplicações. Ela também permite que novos participantes acessem o conteúdo de forma dinâmica, sem a necessidade de planejamento prévio.

Além disso, a solução também apresenta a vantagem de oferecer uma arquitetura escalável para a disseminação de dados, herdada do ContextNet [David et al. 2012]. Esse tipo de escalabilidade é importante para que se consiga dar suporte a cenários com um grande número de clientes, onde muitas vezes um mesmo usuário possui diversas instâncias de produtores e consumidores com configurações diferentes. Outra propriedade importante da solução é a sua compatibilidade com dispositivos móveis, dado que as tecnologias utilizadas e desenvolvidas são compatíveis com o sistema operacional Android ¹.

Toda a comunicação e API do *ContextNet* é encapsulada pelos componentes que são expostos pela API da VideoLib. Isso evita que desenvolvedores precisem ter conhecimento específico sobre as particularidades do desenvolvimento de aplicações para o *middleware* e permite que eles foquem nas funcionalidades que irão resolver os seus problemas.

¹<https://www.android.com/>

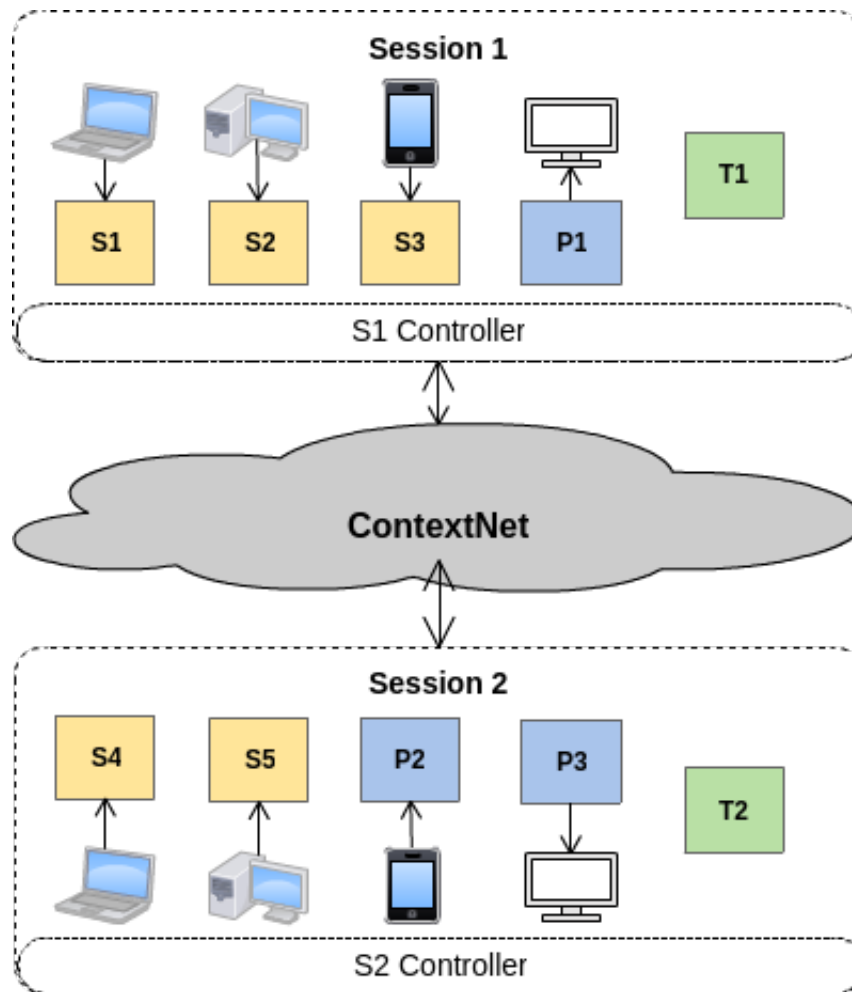


Figura 4.2: Visão geral do funcionamento da VideoLib.

Em relação à arquitetura, a VideoLib é dividida em quatro tipos de componentes principais. Cada um desses componentes deve ser, obrigatoriamente, atrelado a uma sessão. Nesse contexto, uma sessão de compartilhamento de dados pode ser definida como o conjunto de componentes de uma aplicação que estão conectados de forma transparente através do *middleware* proposto e os fluxos de dados que passam por eles. Componentes de uma sessão interagem livremente entre si, mas não têm contato com componentes de outras sessões. A Figura 4.2 mostra uma visão geral do sistema em funcionamento. A seguir são enumerados e descritos os componentes da VideoLib:

- **Fonte (Source):** Componente que irá alimentar o sistema com fluxos multimídia. Ele é responsável por gerar o conteúdo, empacotá-lo no formato de mensagens do *ContextNet* e encaminhá-lo para a rede. É possível enviar comandos de configuração

à fonte. Isto possibilita que sejam selecionadas, dentre uma lista pré-definida de possibilidades, características do vídeo como seu tamanho, formato e taxa de bits.

- **Reprodutor (*Player*):** Responsável pela exibição do conteúdo, ele irá receber o fluxo enviado por alguma fonte, desempacotá-lo e encaminhá-lo para um *player* de vídeo. Analogamente às fontes, é possível se obter dos reprodutores uma lista de formatos suportados. Isso permitiria que houvesse uma negociação automática dos formatos de mídia enviados pelas fontes para que sejam atendidos o maior número possível de reprodutores.
- **Transformador (*Transform*):** Este componente cria novos fluxos de áudio e vídeo a partir de outros fluxos da mesma sessão que estejam participando, que podem ter sido gerados por fontes ou por outros transformadores. Eles tanto podem aplicar filtros sobre um único fluxo, enviando uma nova versão do mesmo para o sistema, como podem combinar diversos conteúdos de diferentes fontes para gerar uma única saída. O projeto deste componente permite que a integração de diferentes técnicas de processamento de vídeo ocorra de forma transparente. A nível de *API*, a transformação é vista como uma caixa preta que aceita alguns fluxos de entrada para gerar outros fluxos de saída. Exemplos de operações que podem ser efetuadas por este componente sobre fluxos de vídeo são a aplicação de um filtro que altere as suas cores, aplicando um efeito de preto e branco, e o agrupamento de diversos vídeos através da criação de um mosaico com todos eles.
- **Controlador (*Controller*):** Diferente dos outros, este componente não está associado a fluxos de dados específicos. Ele serve para controlar remotamente os outros componentes e, dessa forma, gerenciar o funcionamento da sessão que ele integra. A partir do controlador é possível visualizar todos os componentes conectados à sessão e controlar o seu estado com comandos como *iniciar*, *pausar* e *parar*. Também é através dele que são selecionado quais vídeos serão utilizados como entrada para cada reprodutor ou transformador. De uma forma geral, aplicações que utilizam a *VideoLib* comunicam-se diretamente com o controlador para mandar comandos para os demais componentes conectados à sessão.

Para possibilitar que aplicações externas possam ser facilmente integradas ao sistema,

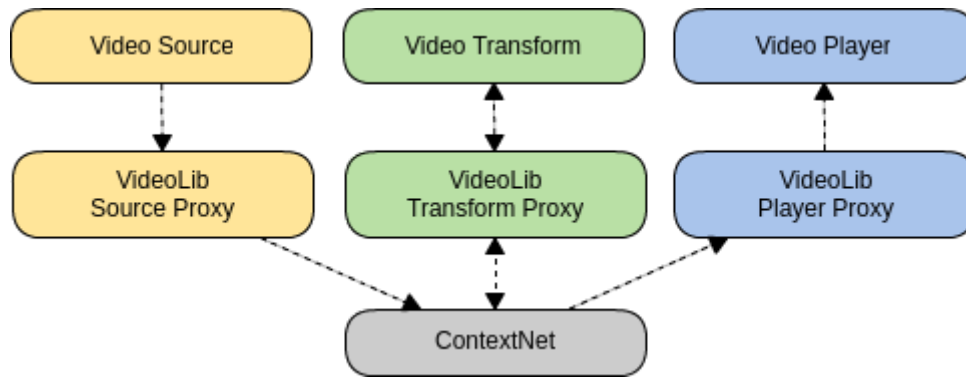


Figura 4.3: Esquema de proxies da VideoLib.

a *VideoLib* prevê o uso de *proxies*. Fontes, destinos e transformadores enviam e recebem os seus respectivos fluxos de dados através desses *proxies*, como demonstrado na Figura 4.3. Para integrar, por exemplo, o reprodutor de vídeo VLC à VideoLib é apenas necessário especializar um componente do reprodutor e implementar alguns métodos como *start*, *stop* e *pause* que mandam comandos específicos para o *software*. Esta estrutura também permite que os *proxies* das fonte ou dos reprodutores convertam os fluxos de vídeo para protocolos diferentes, como o *Digital Living Network Alliance DLNA*[DLNA], muito utilizado em *SmartTVs* ou o *MPEG-DASH*[Sodagar 2011], utilizado em browsers e players específicos.

A criação de transformadores requer várias etapas de configuração. Primeiramente deve-se iniciar o *software* que irá realizar uma transformação com os seus respectivos parâmetros. Posteriormente, deve-se também conectar todas as entradas de vídeo à transformação para que assim o fluxo resultante seja disponibilizado. Para facilitar este processo e permitir que os usuários criem novas transformações remotamente, também foi desenvolvido um **Gerenciador de Transformações (*Transform Manager*)**, que atua como um componente de uma sessão assim como os outros previamente listados, porém sem nenhum fluxo de vídeo passar por ele. Ele define e transmite uma interface de cada transformação que ele disponibiliza e aceita comandos (enviados pelo controlador) para a criação de novas transformações dinamicamente. A Figura 4.4 ilustra esse processo. Nela observa-se que comandos são enviados ao gerenciador requisitando a criação de novas transformação. Ele então instancia os componentes requisitados, que irão interagir com o sistema de forma independente.

Na solução apresentada o maior gargalo para a questão da escalabilidade é a quantidade

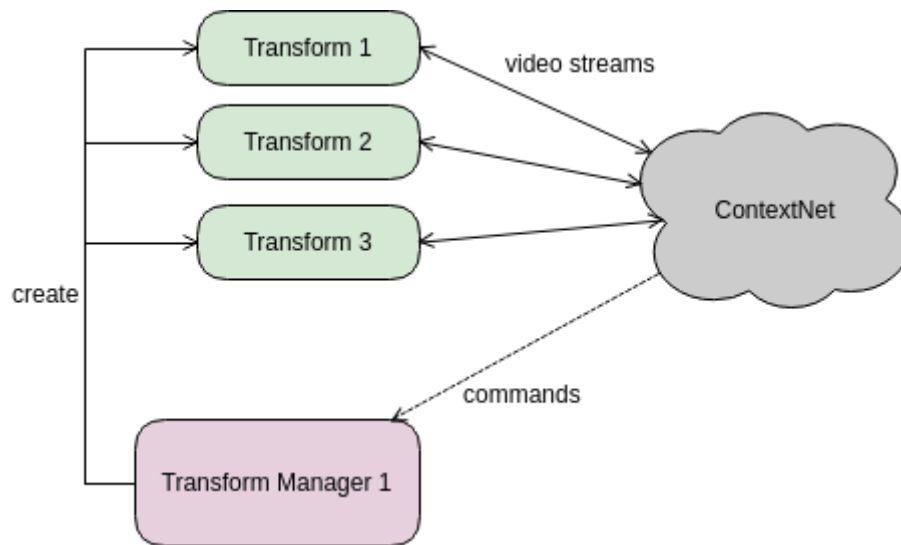


Figura 4.4: Gerenciadores de Transformações.

de nós conectadas a um mesmo ponto de acesso. Dessa forma, a principal forma de escalar a solução é através da adição de novos *gateways* ao sistema. Isto irá balancear o fluxo de dados nos *gateways* de forma que eles não fiquem sobrecarregados. A distribuição dos clientes entre os *gateways* não faz parte do escopo deste trabalho, e atualmente é gerenciada pelo *ContextNet*. A sua otimização poderá ser tratada em trabalhos futuros.

4.2 Metodologia de Desenvolvimento

O desenvolvimento da solução proposta pode ser dividido em algumas etapas, enumeradas abaixo.

- Avaliação inicial da transmissão de vídeo sobre o *ContextNet* e sua posterior otimização para a transmissão de fluxos contínuos de dados.
- Especificação e implementação dos componentes da *VideoLib*.
- Realização de testes das implementações a partir do desenvolvimento de aplicações utilizando a *VideoLib* para explorar suas funcionalidades.
- Realização de testes de desempenho para analisar o comportamento da solução quando submetida a grandes cargas.

- Comparação com algumas das soluções citadas no Capítulo 3.

Detalhes das implementações geradas ao longo do trabalho são mostradas no Capítulo 5 e os resultados obtidos são enunciados e analisados posteriormente, no Capítulo 6.

Capítulo 5

Implementação e Testes

Este capítulo é reservado para a apresentação de detalhes das implementações desenvolvidas no decorrer do desenvolvimento da solução proposta e de suas aplicações de teste. Ele é dividido em seções referentes às etapas listadas na Seção 4.2. Os resultados alcançados por essas implementações serão apresentados e discutidos no capítulo 6.

5.1 Adaptações do ContextNet

O ContextNet foi originalmente concebido para a disseminação de mensagens atômicas entre produtores e consumidores, e não para a transmissão de fluxos de dados. Nesse contexto, ainda no estágio inicial deste trabalho foram feitos estudos sobre a possibilidade de se utilizar o *middleware* para a transmissão de fluxos multimídia. Apesar de requerer alguns ajustes, a transmissão apresentou resultados satisfatórios no que diz respeito a qualidade final do vídeo. Foi verificado que a introdução do *middleware* aumenta a latência e a taxa de dados, mas em contrapartida também observou-se que o ambiente permanece estável mesmo quando há um elevado número de clientes requisitando o vídeo em questão. Uma das alterações necessárias para viabilizar esses testes foi a introdução de uma estrutura de dados para manter os pacotes de vídeo ordenados através de informações de contexto introduzidas nos pacotes do ContextNet [Neto e Silva 2016].

Posteriormente, foi feita outra adaptação no *middleware*. Como já foi discutido na Seção 2.3, o MR-UDP apresenta problemas com transmissões contínuas de fluxos de dados. Devido a essas limitações, foi necessário efetuar a troca deste protocolo. Desenvolveu-se

então um componente com funcionalidades similares, porém baseado no TCP para efetuar as transmissões. De acordo com a Figura 2.3, este novo componente surge como uma opção de suporte para a ClientLib, sobre a qual os nós móveis se comunicam com os *gateways*. Essa troca, entretanto, faz com que algumas propriedades do antigo protocolo sejam perdidas, como a capacidade de lidar com os problemas de conectividade de dispositivos portáteis.

5.2 Desenvolvimento da VideoLib

Toda a comunicação entre os componentes da VideoLib é feita através do ContextNet. Para isso, vários canais de dados são criados para gerenciar as sessões de vídeo, de modo que os usuários consigam se encontrar através do sistema. Cada um desses canais é mapeado para um tópico Pub/Sub diferente, listados na tabela 5.1. Nela podemos observar que existem três tipos diferentes de tópicos. Os tópicos de vídeo são responsáveis pelo transporte dos fluxos gerados por fontes e transformadores. Cada vídeo possui um tópico dedicado exclusivamente a ele. Por outro lado, os tópicos de comando e de sinalização são compartilhados por todos os participantes da sessão. Nos tópicos de comando circulam mensagens de controle destinadas a componentes da sessão. Já nos de sinalização, cada componente envia periodicamente uma mensagem sinalizando aos outros que ainda está ativo.

Tabela 5.1: Tópicos de transmissão de dados do *middleware*

Utilidade	Tópico
Transmissão de vídeo	<i>vlib#vid#{id do componente}</i>
Transmissão de comandos	<i>vlib#com#{sessão}</i>
Sinalização de Componentes Ativos	<i>vlib#sig#{id do componente}</i>

Os três tipos de componentes da VideoLib que possuem contato direto com os fluxos de vídeo são implementados de forma semelhante. Um diagrama de classe desses componentes é mostrado na Figura 5.3

As fontes, primeiramente, são representadas pela classe *VideolibSourceProxy*. Ela trata toda a comunicação com o ContextNet, encapsulando e encaminhando pacotes de vídeo para um canal de dados do *middleware*. Essa classe é então especializada em outra que trata da comunicação desse *proxy* com fontes de vídeo externas. Um exemplo desta especialização

é a classe *VideoLibSourceProxyUDP*. Ela define que os fluxos externos serão recebidos pelo componente via mensagens UDP. Outros protocolos podem ser facilmente integrados através de novas especializações da classe. Por fim, especializa-se a classe de comunicação com o ambiente externo em outras que consigam lidar diretamente com aplicações específicas que fazem a captura ou geração do vídeo em si, como o VLC ou o FFMPEG. No diagrama da Figura 5.3 esta classe é a *MiniSourceProxy*, que controla uma fonte de vídeo simplificada desenvolvida para fins de teste.

Os outros componentes possuem funcionamento análogo. No que diz respeito aos reprodutores, existe uma classe chamada *VideolibPlayerProxy* que recebe pacotes de vídeo, os desencapsula do formato de mensagens do *middleware* e encaminha-os para um reprodutor externo. Ela é especializada em outras classes, assim como as fontes, que tratam do protocolo de envio dessas mensagens para o *player* que irá reproduzir os fluxos. Por fim, uma última especialização da classe é feita para gerenciar características específicas de cada um dos reprodutores de vídeo externos utilizados, como o VLC, por exemplo.

Os transformadores, por sua vez, agregam funcionalidades presentes tanto nas fontes como nos reprodutores. A classe *VideolibTransformProxy* trata tanto da recepção de um número variável de fluxos de mídia como do reenvio de um fluxo transformado para o sistema. Assim como os outros componentes ela pode ser especializada em outras classes que tratam das particularidades dos protocolos de comunicação utilizados pelo componente e, por fim, adaptada para fins específicos de aplicações externas.

Tem-se então o controlador. É ele que oferece ao usuário final a possibilidade de gerenciar os fluxos de áudio e vídeo de uma sessão. Ele se inscreve no canal de sinalização de sua sessão e, conseqüentemente, consegue listar em tempo de execução todos os outros componentes que estão conectados à sessão. Ele também é o principal *publisher* do canal de comandos, podendo dessa forma enviar mensagens para os outros componentes e controlar o seu funcionamento. Alguns dos comandos oferecidos são os atos de iniciar, pausar e parar as transmissões dos componentes, junto com os de conectar fontes, transformadores e reprodutores. Suas funcionalidades estão abstraídas na classe *VideoLibEndpoint*, cujo diagrama de classe é mostrado na Figura 5.4.

É através do controlador que é oferecida para os desenvolvedores de aplicações uma interface fácil para o gerenciamento de aplicações de vídeo. No Código-Fonte 5.1, por

exemplo, são mostradas as etapas que devem ser seguidas para conectar todos os reprodutores de uma sessão a uma única fonte de vídeo.

Código Fonte 5.1: Exemplo de código do controlador

```
1 private void connect(String gatewayIp, int gatewayPort, String sessionId) {  
2  
3     VideolibEndpoint vlibEndpoint = new VideolibEndpoint(gatewayIp, gatewayPort, sessionId);  
4  
5     List<VideolibSource> s = vlibEndpoint.getSources();  
6  
7     VideolibPlayer p : vlibEndpoint.getPlayers() {  
8         p.getVideoFrom(s.get(0).getUid());  
9     }  
10  
11 }
```

Por fim, tem-se o Gerenciador de Transformações, que viabiliza a criação de novas operações de transformação pelos usuários de aplicações da *VideoLib*. Primeiramente, as operações disponibilizadas por este componente são definidas em um arquivo XML. Neste arquivo são descritos as entradas, saídas e os parâmetros necessários para a criação de novos fluxos de vídeo transformado. Um exemplo deste arquivo pode ser encontrado no Apêndice B. O Gerenciador processa este arquivo e disponibiliza a lista de operações suportadas através do ContextNet no tópico de sinalização. Ao receber estas informações, o controlador pode então enviar comandos de criação para o gerenciador. Para facilitar o envio desses comandos foi também desenvolvido um utilitário que, a partir da especificação de uma transformação, requisitando do usuário os valores de cada parâmetro listado. Ao receber o comando de criação, o gerenciador instancia um novo processo responsável pela transformação e seus respectivos *proxies*, que irão se comunicar de forma independente com os outros componentes da sessão.

5.3 Testes Funcionais

Para testar o funcionamento e usabilidade da solução proposta, foi desenvolvida uma aplicação baseada no controlador da *VideoLib* que gerencia os diversos componentes do sistema. Essa aplicação oferece ao usuário uma lista de *Fontes*, *Reprodutores* e *Transformadores* conectados a uma sessão, permitindo que sejam direcionados os fluxos

de vídeo provenientes deles.

Quanto ao desenvolvimento da aplicação, apenas um método precisa ser chamado para efetuar a conexão, por exemplo, entre uma *Fonte* e um *Reprodutor*, de forma similar a apresentada na linha 8 do Código Fonte 5.1. Para o desenvolvedor, toda a configuração dos componentes e endereços de rede se torna transparente. Apenas sabendo o nome da sessão em que deseja se conectar é possível escolher, dentre todos os vídeos disponíveis na sessão (sejam eles provenientes de *Fontes* ou *Transformadores*), o vídeo que cada *Reprodutor* irá exibir. Dessa mesma forma também é possível definir quais fluxos servirão de entrada para uma transformação, que se encarregará do processamento do vídeo.

Algumas transformações foram implementadas para testar a solução. Uma delas foi a operação de "*chroma key*". Para efetuar esta operação em tempo real, o usuário da aplicação precisa apenas conectar duas *Fontes* de vídeo a um *Transformador* e depois ligar a sua saída a um *Reprodutor*. Sem o auxílio da *VideoLib*, a aplicação desta técnica em tempo real precisaria de diversas etapas de configuração de endereços de rede e de formatos de vídeo, além de conhecimento especializado de técnicas e bibliotecas de manipulação de vídeo. A técnica do *chroma key* pode ser facilmente utilizada para inserir diversos participantes remotos em um mesmo espaço virtual, possibilitando a sua interação e criando um ambiente de telepresença [Walker e Sheppard 1999].

Outra transformação implementada foi a criação de um mosaico em tempo real. A partir de várias *Fontes* conectadas a uma mesma sessão é gerado um mosaico de vídeos. Este mosaico é então utilizado para alimentar o sistema com um novo fluxo, que pode ser exibido em qualquer um dos reprodutores conectados à mesma sessão. Isso permite que um usuário consiga visualizar uma grande quantidade de vídeos dos participantes simultaneamente em um único *player* de vídeo. Essa transformação é importante em cenários onde se é necessário mostrar um panorama dos participantes da sessão. O código para a criação de uma transformação e a para a sua conexão à diversas fontes de vídeo é exemplificado no Apêndice C.

Para habilitar a geração de ambas as transformações descritas nesta seção foi utilizada a ferramenta GStreamer, descrita na Seção 2.5. O *pipeline* de componentes implementado para as transformações de mosaico é mostrado na Figura 5.1. No caso do *chromakey* o pipeline utilizado é similar, porém o componente *videomixer* é configurado de forma diferente para

sobrepor os vídeos.

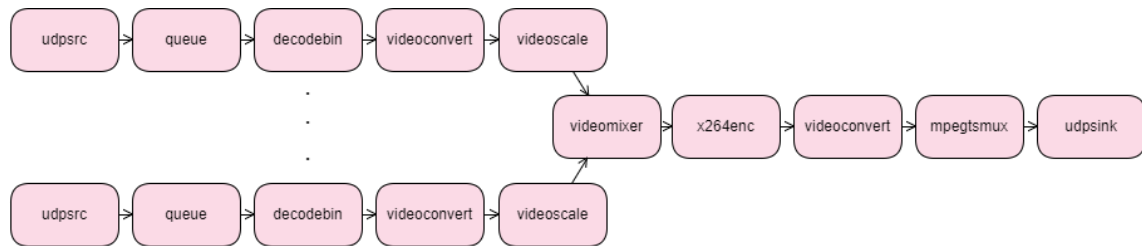


Figura 5.1: Componentes do Gstreamer que formam aplicação do mosaico

5.4 Testes de Desempenho

Também foram realizados testes para analisar o desempenho do sistema e sua capacidade em suportar um grande número de componentes conectados simultaneamente. Nestes testes foram utilizadas máquinas similares com processadores Core i7-6700 e 8GB de memória RAM.

Alguns dos parâmetros avaliados foram a latência das transmissões, que é o tempo que demora para uma imagem chegar ao usuário final, e o número de quadros perdidos em um fluxo de vídeo. A medição desses parâmetros nos permite estimar o atraso introduzido nas transmissões pelo *middleware* e avaliar se há perda de qualidade na transmissão. Os testes foram rodados com números variados de fontes e reprodutores, e foram utilizados 2 vídeos diferentes. Um deles gerado ao vivo, com uma taxa de 1.3Mbps, e o outro originado de um arquivo a uma taxa de 2Mbps.

Para medir a latência, foi capturado o vídeo de um cronômetro com precisão de milissegundos. Este vídeo, ao chegar no destino final, foi exibido em um reprodutor. Em qualquer momento de tempo, a latência da transmissão foi considerada como a diferença entre o valor atual do cronômetro e o valor exibido pelo reprodutor. Esta abordagem pode ser influenciada por alguns fatores externos como a taxa de atualização dos monitores utilizados, a taxa de quadros do cronômetro e a taxa de quadros capturados pela câmera. Isso implica no aumento das margens de erro do resultado, mas permite que seja analisada a variação dos valores com a mudança dos números de fontes e reprodutores. As informações de perda de quadros foram coletadas através do *software* VLC.

Em outra etapa de testes foi analisado o desempenho da operação de mosaico. Foram avaliadas a capacidade de colocar um grande número de vídeos em um mesmo mosaico e a latência na geração do mesmo. Também foi avaliada a possibilidade de aninhar várias transformações desse tipo, gerando um mosaico com um número maior de participantes, como é esquematizado na Figura 5.2

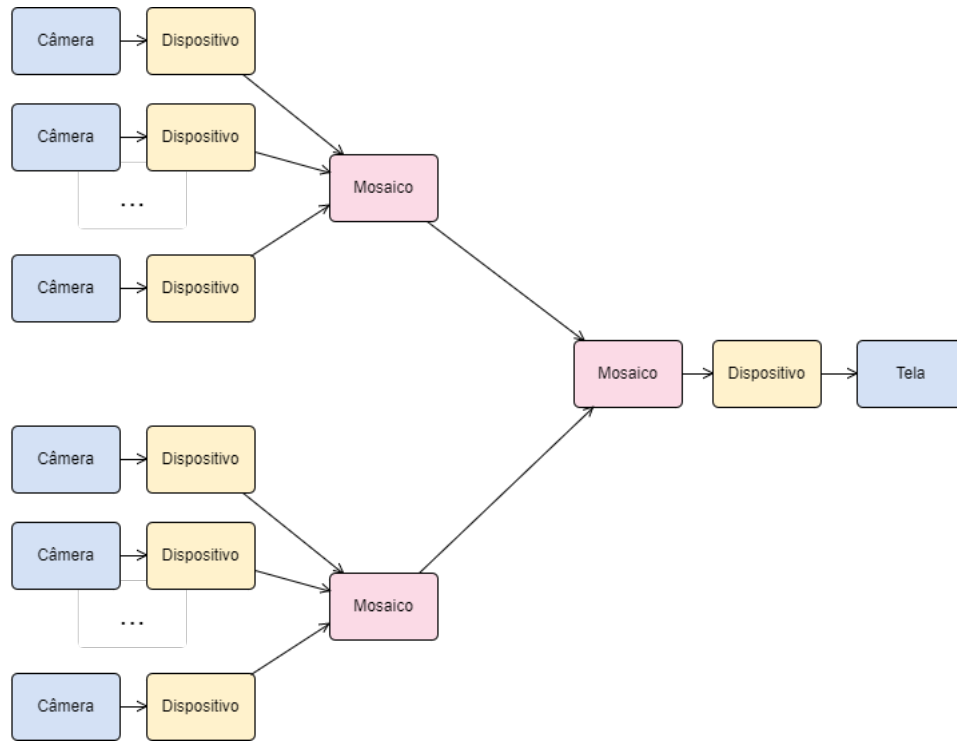


Figura 5.2: Geração aninhada de mosaicos

A latência da geração do mosaico foi medida de maneira similar às transmissões, através da captura e processamento de um vídeo contendo um cronômetro. Calculando a diferença entre o tempo de transmissão de um vídeo (Δ_{tv}) e o tempo de transmissão de um mosaico de vídeos (Δ_{tm}), acha-se a latência introduzida no sistema pelo mosaico (Δ_m). A latência introduzida pelo aninhamento de mosaicos (Δ_{nm}), por sua vez, é estimada multiplicando-se o número de níveis de aninhamento pelo valor de Δ_m .

$$\Delta_m = \Delta_{tm} - \Delta_{tv}$$

$$\Delta_{nm} = n_{niveis} * \Delta_m$$

5.5 Mega-conferência

Como uma outra etapa na validação do *middleware* desenvolvido foi feito um estudo de caso onde múltiplos participantes de uma sessão de videoconferência seriam exibidos em apenas um fluxo de vídeo. Para isso foi utilizado o processo de geração aninhada de mosaicos descrito na Seção 5.4. Para este teste foram utilizadas 5 máquinas distintas com as seguintes configurações:

- 3 máquinas do Tipo A (que serão chamadas de A1, A2 e A3), com processadores Core i7-6700 e 8GB de memória RAM.
- 1 máquina do Tipo B, com processador Core i7-3610QM e 8GB de memória RAM.
- 1 máquina do Tipo C, com processador Core i5-5250U e 8GB de memória RAM.

Na máquina C foi executado um *Gateway*, que serviu como ponto de acesso para os outros componentes do teste. A máquina B serviu como ponto de entrada de vídeos para o sistema e também como ponto de exibição do resultado final. Nas máquinas A1 e A2 foram executadas transformações de mosaicos que condensavam 64 fluxos de vídeo em um. Por fim, na máquina A3 foi executada outra transformação de mosaico que teve como entrada dois vídeos simples e os dois mosaicos gerados pelas máquinas A1 e A2. Dessa forma, foram exibidos 130 participantes de uma conferência em um único fluxo de vídeo, que pode ser facilmente exibido por qualquer reprodutor que esteja integrado ao sistema. Imagens e detalhes da performance das máquinas serão apresentados no Capítulo 6.



Figura 5.3: Diagrama de Classe dos componentes da VideoLib.



Figura 5.4: Diagrama de classes do controlador da VideoLib.

Capítulo 6

Resultados

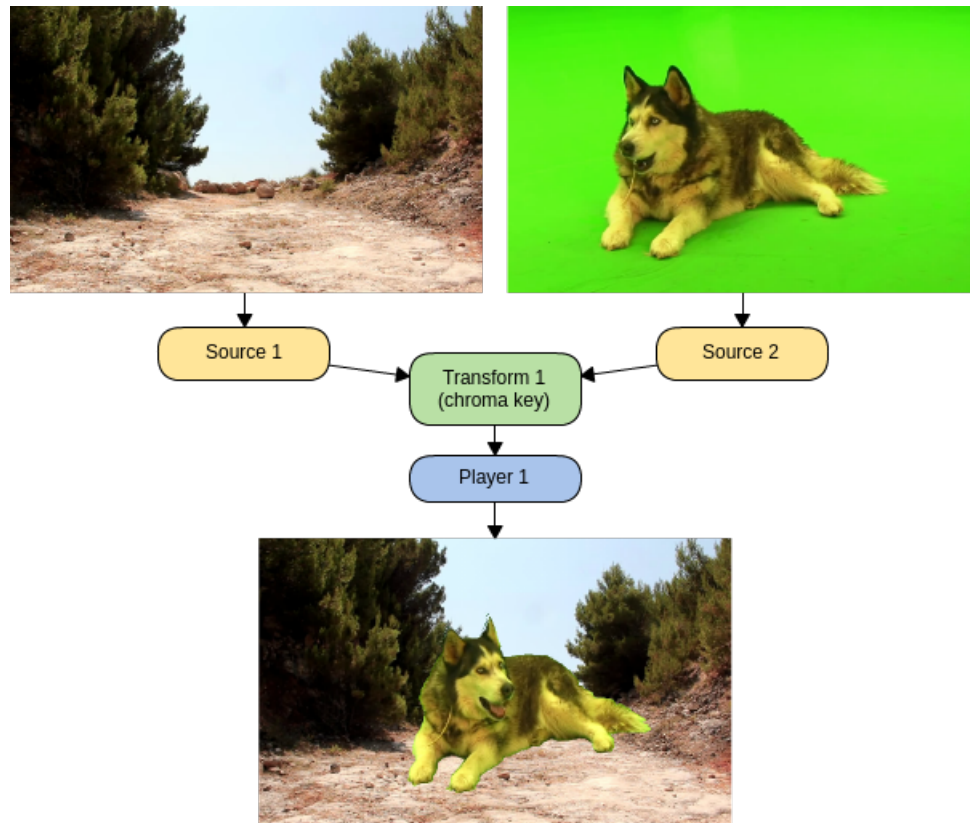
Neste capítulo serão apresentados os resultados obtidos pelas implementações e testes apresentados no capítulo anterior. Por fim, uma vez ilustrados os resultados e o seu desempenho, serão feitas comparações com algumas das soluções discutidas no Capítulo 3.

6.1 Funcionalidades

A primeira parte dos testes realizados para avaliar a solução proposta focou na implementação e uso de componentes diversos para analisar o seu funcionamento. Esses componentes, responsáveis pela captura, transformação e reprodução de fluxos de vídeo foram então conectados por uma aplicação de controle e os resultados são mostrados a seguir.

A Figura 6.1 mostra a operação de *chroma-key* sendo realizada em tempo real. Neste cenário, vídeos de arquivos são transmitidos para instâncias dos componentes fonte da VideoLib e então encaminhados para uma instância do componente responsável pela transformação. Uma vez que a operação é concluída, o vídeo resultante é transmitido para uma instância de um reprodutor. De forma similar, a Figura 6.2 ilustra a operação de criação de mosaico através da VideoLib.

Esses resultados demonstram que a solução implementada funciona e facilita o processo de desenvolvimento de aplicações multimídia. Uma vez implementados os componentes, basta implementar a lógica específica da aplicação junto ao controlador para se obter uma aplicação funcional.

Figura 6.1: Transformação de *chroma key* ao vivo

6.2 Desempenho

Os resultados da primeira parte dos testes, que avalia a latência e a perda de quadros em diferentes situações, podem ser observados na Tabela 6.1. São variados os números de reprodutores e fontes e as taxas dos vídeos transmitidos. Na tabela também estão registradas as taxas totais de entrada e de saída de dados do *middleware*.

Uma das primeiras conclusões tiradas a partir dessa tabela é a de que o número de fontes não tem grande influência na latência das transmissões. Nas três primeiras linhas é mostrado que mesmo quando há uma grande variação na taxa de dados de entrada do sistema a latência permanece a mesma. Já o valor que aparenta ter mais influência na medição da latência é a taxa de dados de saída. Como pode ser observado na Figura 6.3, quanto maior este valor maior tende a ser o atraso observado na entrega dos fluxos. Ela não é o único fator que influencia esse atributo, entretanto. A última linha da tabela mostra que o número de reprodutores também tem alguma influência, pois quando este número sobe muito, mesmo com uma taxa total de dados ainda baixa, a latência começa a sofrer alterações. A perda de

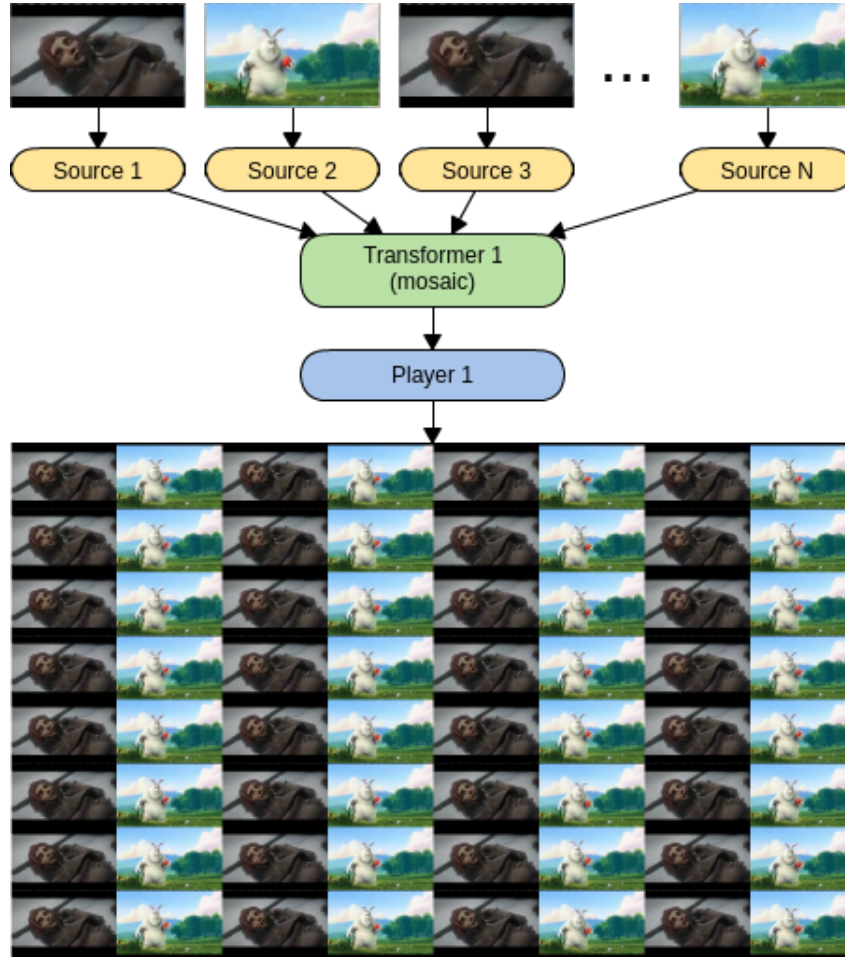


Figura 6.2: Transformação de mosaico ao vivo

quadros apenas é afetada em situações extremas, quando um grande volume de dados está circulando pelo sistema.

No que diz respeito aos testes de mosaico, observou-se que o transformador conseguiu operar satisfatoriamente na geração de murais de alta definição (1920x1080) com até 64 vídeos. Para aumentar este valor, seria necessária a geração aninhada de mosaicos, descrita no capítulo anterior. Em termos de latência, os tempos encontrados são mostrados abaixo.

$$\Delta_{tv} = 1.79$$

$$\Delta_{tm} = 9.56$$

$$\Delta_{nm} = n_{niveis} * 7.77$$

Percebe-se que a introdução da geração de mosaicos aumenta consideravelmente a latência da transmissão. Cada nível de mosaicos acrescenta no sistema um retardo estimado

Tabela 6.1: Latência e taxa de quadros perdidos para transmissões pela *VideoLib*

#Reprodutores	#Fontes	Taxa de Saída (mbps)	Taxa de Entrada (mbps)	Latência (s)	Quadros Perdidos (quadros/minuto)
1	1	1.3	1.3	1.79	0
1	64	1.3	127.3	1.79	0
1	256	1.3	511.3	1.79	0
64	1	83.2	1.3	1.79	0
256	1	332.8	1.3	1.85	0
256	64	511.3	127.3	1.85	0
256	256	511.3	511.3	1.85	0
384	1	499.2	1.3	1.89	0
448	1	582.4	1.3	1.91	90
512	1	665.6	1.3	2.26	250
512	1	51.2	0.1	1.79	0
1024	1	102.4	0.1	1.85	0

em 7.7 segundos, diferença entre os valores de Δ_{tv} e Δ_{tm} . Assim sendo, este recurso deve ser utilizada apenas quando o retardo não for crucial para o bom funcionamento da aplicação.

Os testes realizados indicam que a solução proposta herda do *ContextNet* a escalabilidade também para a disseminação de vídeo. Alguns casos de teste apresentados mostraram que o *middleware* permanece estável mesmo com um grande número de participantes e altas taxas de entrada e saída de dados. Os transformadores também mostraram um bom funcionamento, apesar de em alguns casos introduzirem valores elevados de latência.

6.3 Análise Comparativa

Uma forma de comparar a solução proposta com as soluções de transmissão de vídeo em tempo real descritas na Seção 3.2 é através da modelagem dos cenários em que elas são utilizadas a partir da *VideoLib*.

A Figura 6.4 mostra um cenário similar aos experimentos de Ubik, modelados na

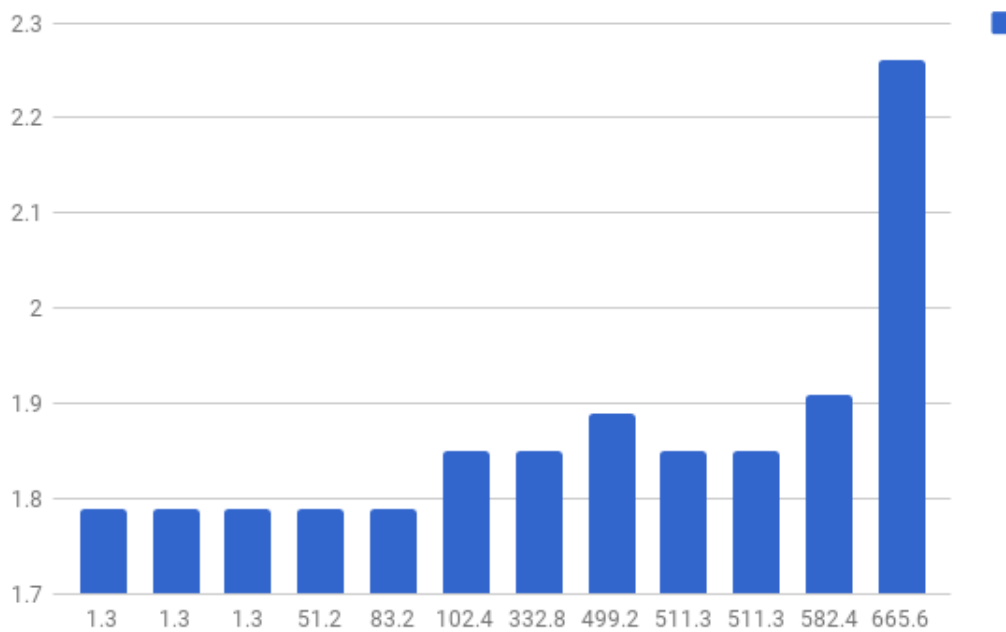


Figura 6.3: Variação da latência com a taxa de saída

Figura 3.2. Nela, pode-se ver que cada uma das câmeras é atrelada a uma instância de uma fonte e cada uma das telas a um componente de reprodução. A vantagem dessa modelagem é que cada um desses componentes apenas precisa se conectar a um *gateway* do ContextNet. Uma simples instância de um controlador seria suficiente para conectar todos os componentes e direcionar os fluxos para seus destinos. Na forma como Ubik modela seus cenários é necessária a configuração das transmissões com os endereços de origem e destino de um dos fluxos. Também é uma vantagem da VideoLib a disponibilização das transformações, facilitando a combinação dos vídeos de várias bailarinas para gerar um único através da operação de *chromakey*.

A Figura 6.5, por sua vez, é referente ao cenário de uso do Arthron para a transmissão de cirurgias em um hospital, que foi ilustrado na Figura 3.5. Uma das maiores diferenças entre as modelagens dos cenários é que o Arthron conecta as fontes e reprodutores diretamente, enquanto no caso do ContextNet a distribuição de dados é feita através do modelo Pub/Sub. Isso mostra a escalabilidade da solução, pois o aumento do número de produtores e consumidores não acarretará na sobrecarga dos componentes. Isso também possibilita que sejam implementadas algumas funcionalidades a mais nesse cenário, como por exemplo a

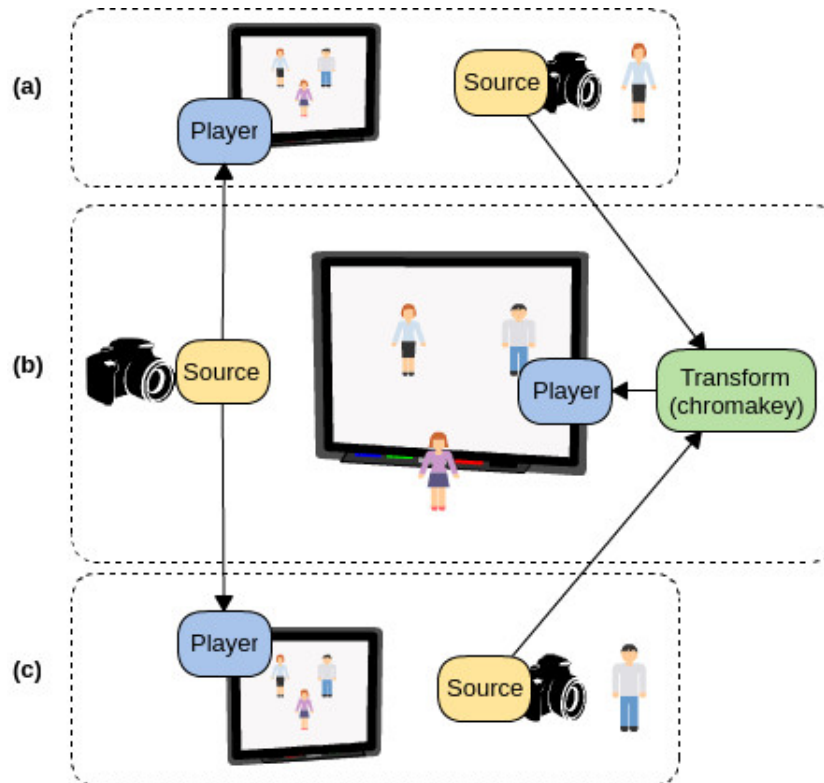


Figura 6.4: Performance telemática utilizando a VideoLib.

disponibilização desses fluxos para alunos em seus *smartphones*.

6.4 Mega-conferência

O estudo de caso da mega-conferência teve como objetivo a inclusão de um grande número de participantes distintos em apenas um vídeo, permitindo que qualquer reprodutor capaz de tocar um vídeo HD possa ser espectador desta conferência. Foram avaliados o comportamento das máquinas envolvidas através da medição das taxas de entrada e saída de dados na rede e da porcentagem de uso de CPU de cada máquina. Para a análise do uso de CPU das máquinas deve-se levar em consideração que elas possuem processadores diferentes, descritos no Capítulo 5.

No gráfico “a” da Figura 6.6 pode-se observar que mesmo com um alto fluxo de dados passando por um *gateway* (15MB/s), o uso de CPU permanece estável ao longo do tempo. No gráfico “b” observa-se que o envio de vídeos não requer tanto processamento da máquina utilizada, e que esta medida apenas se eleva quando se está exibindo um vídeo de fato.

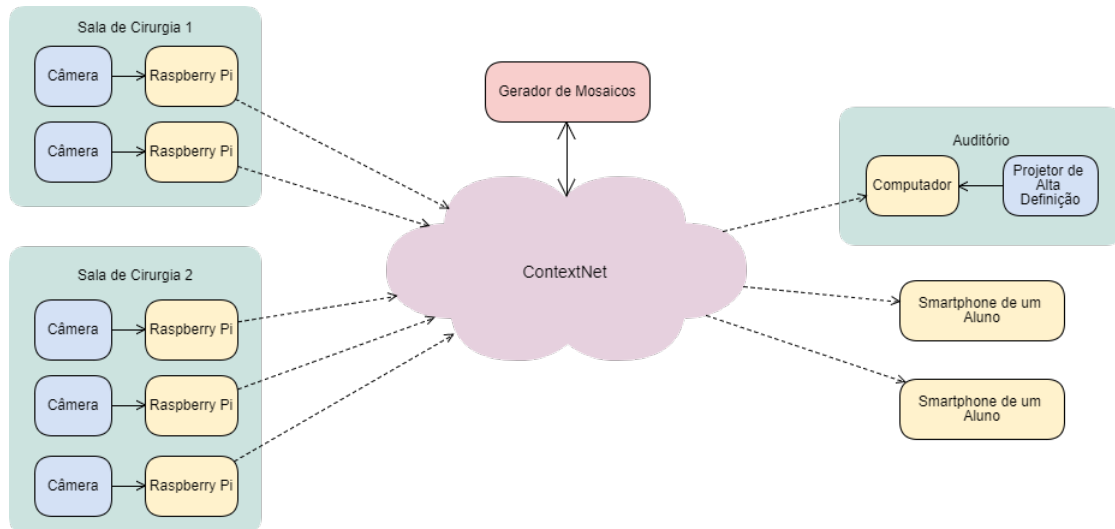


Figura 6.5: Modelagem do cenário do caso de uso

Nos outros três gráficos percebe-se que a geração de mosaicos é uma tarefa mais custosa em termos de recursos computacionais, tendo em vista que é necessária a decodificação e codificação de diversos fluxos de vídeo.

O resultados destes testes pode ser visto na Figura 6.7. Números foram sobrepostos aos vídeos para ilustrar que estes tratam-se de fluxos multimídia diferentes e independentes entre si.

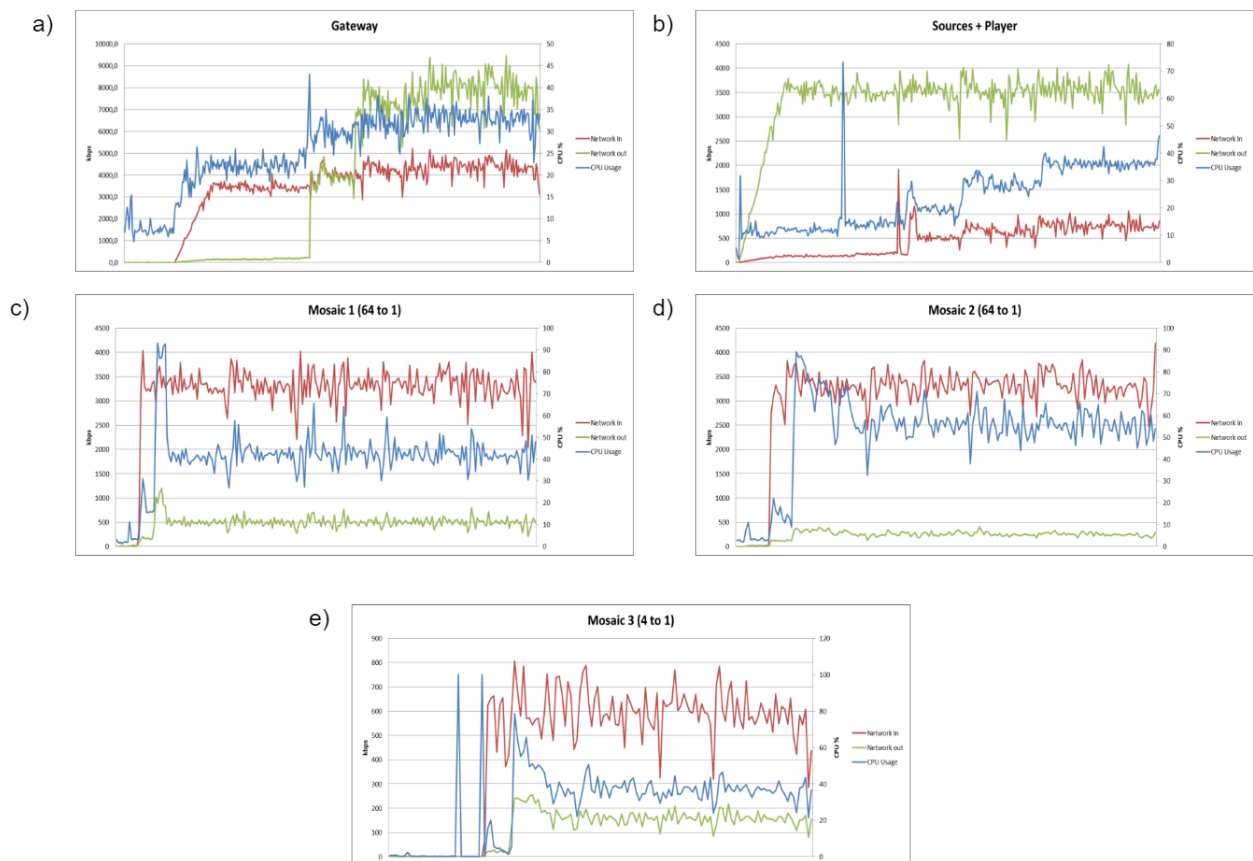


Figura 6.6: Utilização de recursos das máquinas da mega-conferência



Figura 6.7: Fluxo final do cenário de uso da mega conferência

Capítulo 7

Criando Uma Aplicação com a VideoLib

Um dos objetivos deste trabalho é a simplificação das etapas de desenvolvimento de aplicações multimídia. Retirando a responsabilidade do processamento e da distribuição de fluxos multimídia dos desenvolvedores, estes precisariam apenas se preocupar com o desenvolvimento da lógica específica de suas aplicações. Para isso, a *VideoLib* deve se mostrar suficientemente flexível, no sentido de atender as necessidades dos mais variados ambientes para os quais suas aplicações serão desenvolvidas. Este capítulo tem como objetivo exemplificar o processo de desenvolvimento de aplicações para a *VideoLib* através de uma aplicação exemplo que utiliza algumas das funcionalidades providas pelo *middleware*.

7.1 A aplicação

A aplicação apresentada neste capítulo consiste em um visualizador de vídeos de uma sessão de compartilhamento de conteúdo multimídia. Ilustrada na Figura 7.1, ela possui uma área ocupada por um *player* de vídeo principal e outros três reprodutores auxiliares de menor tamanho. Ela também possui um painel de controle responsável por listar os fluxos disponíveis no sistema e encaminhá-los para uma de suas telas. Ainda no painel, a aplicação também exibe uma lista de transformações disponíveis para o usuário e permite que ele crie novas transformações de acordo com as suas necessidades.

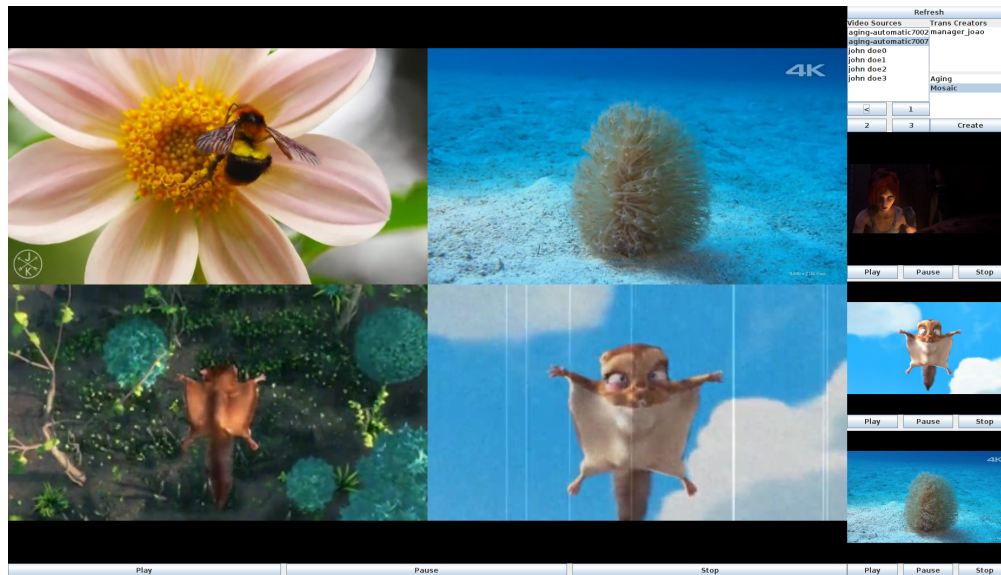


Figura 7.1: Visão geral da aplicação.

7.2 Desenvolvimento

A aplicação foi desenvolvida utilizando a biblioteca *Swing* do ambiente Java para a criação de interfaces gráficas e o pacote *vlcj*, que oferece um reproduutor de vídeo compatível com as interfaces do *Swing*. Esta seção irá tratar apenas das etapas de desenvolvimento que envolvem a *VideoLib*, uma vez que a utilização destas ferramentas está fora do escopo deste trabalho. Todas as funções referentes ao *middleware* exibidas nesta seção estão descritas no Apêndice A.

Como primeiro passo no desenvolvimento deve-se instanciar os componentes necessários para o funcionamento da aplicação. Será necessário um controlador, representado por um objeto da classe *VideolibEndpoint* e três *players*, representados pela classe *MiniDecoderProxy*. A criação destes componentes é mostrada no Código Fonte 7.1 nas linhas 2 e de 4 a 7, respectivamente.

Posteriormente, deve-se preencher as listas de fontes de vídeo e de gerenciadores de transformações. Estas listas são obtidas através de métodos do controlador, utilizados nas linhas 7 e 8 do Código Fonte 7.2. Ao se selecionar um gerenciador, deve-se preencher a lista de transformações que ele disponibiliza para o usuário. A configuração desta lista é feita pelo método mostrado nas linhas 1 a 4 do mesmo código.

Por fim, tem-se o código referente à configuração dos botões do painel de configuração.

Código Fonte 7.1: Criando componentes da *VideoLib*

```

1 private void setupCnet() {
2     vlibEndpoint = new VideolibEndpoint(gatIp, gatPort, session);
3
4     dec0 = new MiniDecoderProxy(gatIp, gatPort, name + "main", session, localhost, port[0]);
5     dec1 = new MiniDecoderProxy(gatIp, gatPort, name + "mini1", session, localhost, port[1]);
6     dec2 = new MiniDecoderProxy(gatIp, gatPort, name + "mini2", session, localhost, port[2]);
7     dec3 = new MiniDecoderProxy(gatIp, gatPort, name + "mini3", session, localhost, port[3]);
8 }

```

Código Fonte 7.2: Configurando as listas de componentes

```

1 private void setTransformListFromManager(ListSelectionEvent e) {
2     VideolibTransformManager tManager = getSelectedTransfManager();
3     setupListContent(transfList, tManager.getConfig().getTransformSpecs());
4 }
5
6 private void setupListsContent() {
7     setupListContent(sourceList, vlibEndpoint.getStreamSources());
8     setupListContent(transfManagerList, vlibEndpoint.getTransformManagers());
9     transfManagerList.addListSelectionListener(this::setTransformListFromManager);
10 }

```

Nas linhas 1 a 8 do Código Fonte 7.3 estes botões são mapeados para chamadas de métodos da aplicação. Ao se clicar em algum dos 4 botões abaixo da lista de fontes, o vídeo referente à fonte selecionada será exibido em uma das telas, dependendo de qual botão foi clicado. Para isso, é executado o método "*getVideoFrom()*", como é mostrado nas linhas 10 a 14 do mesmo código. Ao clicar em "*Create*", abaixo da lista de transformações, a aplicação cria uma janela, disponibilizada pela *VideoLib*, que se encarrega de recolher as informações necessárias para a criação da transformação em questão e de enviar a mensagem de criação para o gerenciador selecionado. O código responsável por este processo está nas linhas 16 a 20. Ao atualizar a lista de fontes, a nova fonte de vídeo correspondente à transformação será mostrada junto às outras. As etapas descritas neste parágrafo estão ilustradas na Figura 7.2

7.3 Cenário de Uso

A aplicação descrita neste capítulo pode ser utilizada em diversos cenários. Um deles é o da telemedicina, já mencionado neste trabalho. Câmeras compactas atreladas a dispositivos

Código Fonte 7.3: Configurando botões da aplicação

```

1  private void setupButtons() {
2      // ...
3      _0Button.addActionListener(actionEvent -> setVideoSourceFromList(0));
4      _1Button.addActionListener(actionEvent -> setVideoSourceFromList(1));
5      _2Button.addActionListener(actionEvent -> setVideoSourceFromList(2));
6      _3Button.addActionListener(actionEvent -> setVideoSourceFromList(3));
7      createTransformButton.addActionListener(actionEvent -> createTransform());
8  }
9
10 private void setVideoSourceFromList(int i) {
11     VideolibComponent selectedSource = getSelectedSource();
12     getDecoderFromIndex(i).getVideoFrom(selectedSource.getUid());
13     // ...
14 }
15
16 private void createTransform() {
17     VideolibTransformManager tManager = getSelectedTransfManager();
18     TransformsConfig.TransformSpec transform = getSelectedTransform();
19     new TransformCreatorDialog(tManager, transform, vlibEndpoint.getStreamSources());
20 }

```

Raspberry Pi seriam distribuídas por todas as salas de um centro cirúrgico de forma não intrusiva, capturando imagens do paciente, dos monitores de sinais vitais, do anestesista, das ferramentas utilizadas e de outras localizações chave para as diversas atividades que ocorrem durante cirurgias. Estas imagens seriam então enviadas via WiFi para computadores que estariam executando *gateways* do ContextNet. Também poderiam haver máquinas executando gerenciadores de transformações dedicados a gerar mosaicos com todos os vídeos de uma cirurgia específica para uma visualização mais completa do ambiente. A qualquer momento, um usuário com acesso à rede interna do hospital poderia, então, se conectar a essa instância do ContextNet e consumir estes vídeos a partir da aplicação. Professores poderiam utilizá-la para exibir os procedimentos para os seus alunos e gerar novas transformações a partir dos vídeos. Além disso, os próprios alunos poderiam ter acesso a essas transmissões para estudar em seu tempo livre.

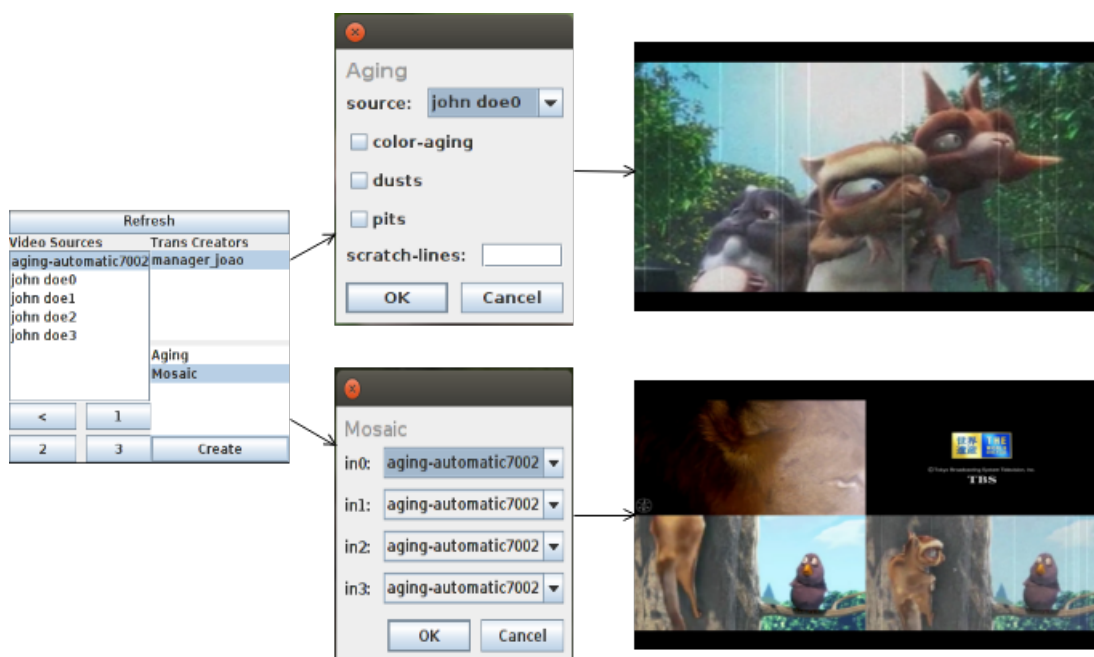


Figura 7.2: Processo de criação de Transformações.

Capítulo 8

Conclusão

Neste trabalho foi apresentada uma solução para a distribuição escalável de conteúdo multimídia em tempo real. Ela tem como propriedades a facilidade de configuração, a escalabilidade e a adaptabilidade a um grande número de dispositivos e de cenários de aplicação.

Em termos de qualidade do vídeo, os resultados obtidos mostram que o sistema conseguiu manter uma transmissão sem perdas de conteúdo na maior parte do tempo com uma boa quantidade de clientes. Pode-se aumentar essa quantidade através da adição de mais máquinas na rede que suporta o *middleware* utilizado. Em termos de latência, entretanto, houve um certo retardo introduzido nas transmissões, principalmente ao se utilizar componentes que geram novos fluxos de vídeo a partir de outros já existentes. Esse retardo deve ser observado cuidadosamente ao aplicar a solução em cenários onde o tempo de transmissão é um fator crítico. Em outros cenários, entretanto, esse fator não interfere na aplicabilidade da solução.

Através das mesmas interfaces de *Fontes*, *Reprodutores*, *Transformadores* e *Controladores*, mostrou-se que é possível implementar aplicações que vão desde performances artísticas distribuídas até aplicações de telemedicina. Acredita-se que devido a essas características a VideoLib pode dar suporte ao desenvolvimento de novos tipos de aplicações baseadas na disseminação e produção colaborativa de conteúdos multimídia em tempo real entre muitos participantes. Essas aplicações podem contornar a rigidez dos sistemas comumente disponíveis e beneficiar-se de uma maior quantidade de dispositivos de captura e reprodução para criar modelos colaborativos dinâmicos e distintos dos modelos

tradicionais de aplicação multimídia.

Este trabalho pode servir como base para diversos trabalhos futuros, tanto desenvolvendo melhorias para o *middleware* como construindo aplicações utilizando-o. Alguns exemplos desses trabalhos são:

- Analisar possíveis mudanças estruturais no *middleware* visando a sua otimização, encontrando gargalos para a escalabilidade e integrando partes da API à rede DDS, evitando intermediários (*proxies*) na comunicação entre os componentes.
- Investigação de formas de diminuir a latência encontrada, viabilizando a utilização da solução em cenários conversacionais.
- Desenvolvimento de métodos de controle da posse de voz e de tratamento do áudio quando um número muito grande de usuários estiverem transmitindo vídeos em uma mesma sessão.
- Desenvolvimento de técnicas para sincronizar a visualização dos fluxos do sistema.
- Aplicação de protocolos de segurança para garantir a confidencialidade das informações trocadas através do *middleware*
- Desenvolvimento de um serviço de streaming adaptativo utilizando o *middleware*, dado que o sistema o sistema permite o chaveamento entre diversos pedaços de vídeo de diferentes formatos.

Resultados parciais deste trabalho foram aceitos para publicação como artigos completos no *WebMedia 2016* [Neto e Silva 2016] e *WebMedia 2017* [Neto e Silva 2017].

Bibliografia

- [Adler-Milstein, Kvedar e Bates 2014]ADLER-MILSTEIN, J.; KVEDAR, J.; BATES, D. W. Telehealth among us hospitals: several factors, including state reimbursement and licensure policies, influence adoption. *Health Affairs*, Health Affairs, v. 33, n. 2, p. 207–215, 2014.
- [Al-Madani, Al-Roubaiey e Baig 2014]AL-MADANI, B.; AL-ROUBAIEY, A.; BAIG, Z. A. Real-time qos-aware video streaming: a comparative and experimental study. *Advances in Multimedia*, Hindawi Publishing Corp., v. 2014, p. 1, 2014.
- [Al-Madani, Al-Saeedi e Al-Roubaiey 2013]AL-MADANI, B.; AL-SAEEDI, M.; AL-ROUBAIEY, A. A. Scalable wireless video streaming over real-time publish subscribe protocol (rtps). In: IEEE COMPUTER SOCIETY. *Proceedings of the 2013 IEEE/ACM 17th International Symposium on Distributed Simulation and Real Time Applications*. [S.l.], 2013. p. 221–230.
- [Banavar et al. 1999]BANAVAR, G. et al. An efficient multicast protocol for content-based publish-subscribe systems. In: IEEE. *Distributed Computing Systems, 1999. Proceedings. 19th IEEE International Conference on*. [S.l.], 1999. p. 262–272.
- [Bova e Krivoruchka 1999]BOVA, T.; KRIVORUCHKA, T. Reliable udp protocol. *draft-ietf-sigtran-reliable-udp-00.txt*, 1999.
- [Cisco 2015]CISCO, V. *Forecast and Methodology, 2015-2020*. 2015.
- [David et al. 2012]DAVID, L. et al. A communication middleware for scalable real-time mobile collaboration. In: IEEE. *Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE), 2012 IEEE 21st International Workshop on*. [S.l.], 2012. p. 54–59.

- [David et al. 2013]DAVID, L. et al. A dds-based middleware for scalable tracking, communication and collaboration of mobile nodes. *Journal of Internet Services and Applications*, Springer, v. 4, n. 1, p. 1–15, 2013.
- [Detti et al. 2010]DETTI, A. et al. Streaming h. 264 scalable video over data distribution service in a wireless environment. In: IEEE. *World of Wireless Mobile and Multimedia Networks (WoWMoM), 2010 IEEE International Symposium on a*. [S.l.], 2010. p. 1–3.
- [DLNA]DLNA, D. Overview and vision whitepaper, 2006. URL:< http://www.dlna.org/en/industry/about/dlna_white_paper_2006.pdf.
- [Dworak et al. 2011]DWORAK, A. et al. Middleware trends and market leaders 2011. In: *Conf. Proc.* [S.l.: s.n.], 2011. v. 111010, n. CERN-ATS-2011-196, p. FRBHMULT05.
- [Endler et al. 2011]ENDLER, M. et al. Contextnet: context reasoning and sharing middleware for large-scale pervasive collaboration and social networking. In: ACM. *Proceedings of the Workshop on Posters and Demos Track*. [S.l.], 2011. p. 2.
- [Eugster et al. 2003]EUGSTER, P. T. et al. The many faces of publish/subscribe. *ACM Computing Surveys (CSUR)*, ACM, v. 35, n. 2, p. 114–131, 2003.
- [Filho et al. 2012]FILHO, E. S. et al. Uma ferramenta para gerenciamento e transmissão de fluxos de vídeo em alta definição para telemedicina. *Salão de Ferramenta SBRC–2012*, 2012.
- [García-Valls, Basanta-Val e Estévez-Ayres 2010]GARCÍA-VALLS, M.; BASANTA-VAL, P.; ESTÉVEZ-AYRES, I. Adaptive real-time video transmission over dds. In: IEEE. *Industrial Informatics (INDIN), 2010 8th IEEE International Conference on*. [S.l.], 2010. p. 130–135.
- [Gharai et al. 2006]GHARAI, L. et al. Experiences with high definition interactive video conferencing. In: IEEE. *Multimedia and Expo, 2006 IEEE International Conference on*. [S.l.], 2006. p. 433–436.
- [Halak e Ubik 2009]HALAK, J.; UBIK, S. Mttp-modular traffic processing platform. In: IEEE. *Design and Diagnostics of Electronic Circuits & Systems, 2009. DDECS'09. 12th International Symposium on*. [S.l.], 2009. p. 170–173.

- [Hock e Lingxia 2014]HOCK, K. S.; LINGXIA, L. Automated processing of massive audio/video content using ffmpeg. *Code4Lib Journal*, v. 23, 2014.
- [Holub et al. 2012]HOLUB, P. et al. Ultragrid: Low-latency high-quality video transmissions on commodity hardware. In: ACM. *Proceedings of the 20th ACM international conference on Multimedia*. [S.l.], 2012. p. 1457–1460.
- [Huang e Garcia-Molina 2004]HUANG, Y.; GARCIA-MOLINA, H. Publish/subscribe in a mobile environment. *Wireless Networks*, Springer, v. 10, n. 6, p. 643–652, 2004.
- [Melo et al. 2010]MELO, E. A. G. de et al. Arthron 1.0: Uma ferramenta para transmissão e gerenciamento remoto de fluxos de mídia. 2010.
- [Neto e Silva 2016]NETO, J. Martins de O.; SILVA, L. David Nery e. Video streaming over publish/subscribe. In: ACM. *Proceedings of the 22nd Brazilian Symposium on Multimedia and the Web*. [S.l.], 2016. p. 183–189.
- [Neto e Silva 2017]NETO, J. Martins de O.; SILVA, L. David Nery e. A middleware for the development of distributed collaborative video applications. In: ACM. *Proceedings of the 23rd Brazilian Symposium on Multimedia and the Web*. [S.l.], 2017. p. to appear.
- [Nimmi et al. 2014]NIMMI, S. et al. Real-time video streaming using gstreamer in gnu radio platform. In: IEEE. *Green Computing Communication and Electrical Engineering (ICGCCEE), 2014 International Conference on*. [S.l.], 2014. p. 1–6.
- [OMG 2006]OMG. Data distribution service for real-time systems. *OMG, Tech. Rep. OMG Available Specification*, 2006.
- [Otnes, Eastwood e Colin 2015]OTNES, R.; EASTWOOD, J.; COLIN, M. E. *Using GStreamer for acoustic signal processing in deployable sensor nodes*. [S.l.]: IEEE, 2015.
- [Pardo-Castellote 2003]PARDO-CASTELLOTE, G. Omg data-distribution service: Architectural overview. In: IEEE. *Distributed Computing Systems Workshops, 2003. Proceedings. 23rd International Conference on*. [S.l.], 2003. p. 200–206.

[Rawle et al. 2017]RAWLE, M. et al. Improving education and supervision of queensland x-ray operators through video conference technology: A teleradiography pilot project. *Journal of Medical Radiation Sciences*, Wiley Online Library, 2017.

[RNP 2005]RNP. *RNP lança rede multigigabit com espetáculo de dança interativo*. 2005. http://portal.rnp.br/web/rnp/imprensa/-/rutelistaconteudo/RNP-lanca-rede-multigigabit-com-espetaculo-de-danca-interativo/607787_6Ca1. Accessed: 2017-05-25.

[Santana 2014]SANTANA, I. Networked dance performance: a new temporality. *Liminalities: a Journal of Performance Studies*, Liminalities, v. 10, n. 1, p. 2–19, 2014.

[Silva, Endler e Roriz 2013]SILVA, L.; ENDLER, M.; RORIZ, M. Mr-udp: Yet another reliable user datagram protocol, now for mobile nodes. *Monografias em Ciência da Computação*, nr, v. 1200, p. 06–13, 2013.

[Silva 2014]SILVA, L. D. N. e. *A Scalable Middleware for Structured Data Provision and Dissemination in Distributed Mobile Systems*. Tese (Doutorado) — PhD thesis, PUC-Rio, 2014.

[Sodagar 2011]SODAGAR, I. The mpeg-dash standard for multimedia streaming over the internet. *IEEE MultiMedia*, IEEE, v. 18, n. 4, p. 62–67, 2011.

[Tavares 2015]TAVARES, T. A. Desafios e realizações da iniciativa da rnp: Grupo de trabalho em mídias digitais e artes. *Revista Eletrônica MAPA D2-Mapa e Programa de Artes em Dança (e Performance) Digital*, v. 2, n. 2, 2015.

[Ubik et al. 2016]UBIK, S. et al. Cyber performances, technical and artistic collaboration across continents. *Future Generation Computer Systems*, Elsevier, v. 54, p. 306–312, 2016.

[Walker e Sheppard 1999]WALKER, G.; SHEPPARD, P. Telepresence—the future of telephony. In: *Telepresence*. [S.l.]: Springer, 1999. p. 1–13.

[Wang, Zhang e Ge 2014]WANG, P.; ZHANG, L.; GE, Q. Video surveillance in simplex wireless channel based on gstreamer. In: IEEE. *Advanced Research and Technology in Industry Applications (WARTIA), 2014 IEEE Workshop on*. [S.l.], 2014. p. 378–382.

- [Wen 2008]WEN, C. L. Telemedicina e telessaúde—um panorama no brasil. *Informática Pública*, v. 10, n. 2, p. 7–15, 2008.
- [Xiaohua, Xiuhua e Caihong 2013]XIAOHUA, L.; XIUHUA, J.; CAIHONG, W. Design and implementation of a real-time video stream analysis system based on ffmpeg. In: IEEE. *Software Engineering (WCSE), 2013 Fourth World Congress on*. [S.l.], 2013. p. 212–216.
- [Xie et al. 2013]XIE, Y. et al. A study on the effectiveness of videoconferencing on teaching parent training skills to parents of children with adhd. *Telemedicine and e-Health*, Mary Ann Liebert, Inc. 140 Huguenot Street, 3rd Floor New Rochelle, NY 10801 USA, v. 19, n. 3, p. 192–199, 2013.
- [Xu e Cao 2014]XU, Y. G.; CAO, S. X. Real-time video acquisition and frame compression processing technology based on ffmpeg. In: TRANS TECH PUBL. *Applied Mechanics and Materials*. [S.l.], 2014. v. 631, p. 494–497.
- [Zhao, Zhou e Jin 2015]ZHAO, H.; ZHOU, C.-l.; JIN, B.-z. Design and implementation of streaming media server cluster based on ffmpeg. *The Scientific World Journal*, Hindawi Publishing Corporation, v. 2015, 2015.

Apêndice A

API da VideoLib

VideoLibEndpoint

VideoLibEndpoint(String gateway, int port, String session)

List<VideoLibSource> getSources()

VideoLibSource getSources(String id)

List<VideoLibPlayer> getPlayers()

VideoLibPlayer getPlayers(String id)

List<VideoLibTransform> getTransform()

VideoLibTransform getTransform(String id)

VideoLibSource

void start()

void stop()

void pause()

void quit()

VideoLibPlayer

void start()

void stop()

void pause()

void quit()

void getVideoFrom(String id)

VideoLibTransform

void start()

void stop()

void pause()

void quit()

void getVideoFrom(String id, int position)

Apêndice B

Arquivo de configuração do Gerenciador de transformações

Código Fonte B.1: Exemplo de arquivo de configuração do gerenciador de transformações

```
1 <TransformConfig>
2   <TransformList>
3     <Transform name="Aging" id="0" desc="Ages the image of a video"
4       filename="agingtest.out" ins="1" outs="1"
5     >
6       <in id="0" name="source" desc="Input video stream" />
7       <out id="0" name="sink" desc="Output video stream " />
8       <param
9         name="color-aging"
10        type="boolean"
11        id="0"
12        desc="Should it change the video color?"
13      />
14      <param name="dusts" type="boolean" id="1" desc="Should it become dusty?" />
15      <param name="pits" type="boolean" id="2" desc="Gstreamer unknown param" />
16      <param
17        name="scratch-lines"
18        type="number"
19        max="20"
20        id="3"
21        desc="Number of scratch lines"
22      />
23    </Transform>
24
25    <Transform
26      name="Mosaic" id="1"
27      desc="Creates a mosaic from an arbitrary number of video streams"
```

```
28         filename="mosaictest.out" ins="-1" outs="1">
29     </Transform>
30
31 </TransformList>
32 </TransformConfig>
```

Apêndice C

Código para a criação de uma Transformação

Código Fonte C.1: Criação e inicialização de uma transformação

```
1 package br.ufpb.ci.lavid.videolib.main;
2
3 import java.util.ArrayList;
4 import java.util.List;
5
6 import br.ufpb.ci.lavid.videolib.api.VideolibEndpoint;
7 import br.ufpb.ci.lavid.videolib.api.components.VideolibSource;
8 import br.ufpb.ci.lavid.videolib.proxies.MiniDecoderProxy;
9 import br.ufpb.ci.lavid.videolib.proxies.MiniEncoderProxy;
10 import br.ufpb.ci.lavid.videolib.proxies.MiniTransformProxyN;
11
12 public class MainTransf {
13     public static void main(String args[]) {
14
15         // Coletando par metros de entrada
16         String gatIp = args[0];
17         int nInputs = Integer.parseInt(args[1]);
18         int basePort = Integer.parseInt(args[2]);
19         int finalPort = Integer.parseInt(args[3]);
20
21         ArrayList<String> ips = new ArrayList<>();
22         ArrayList<Integer> ports = new ArrayList<>();
23
24         String transfUid = null;
25
26         // Criando lista de IPs e Portas para onde ir o os v deos
27         for (int i = 0; i < nInputs; i++) {
```

```
28     ips.add("127.0.0.1");
29     ports.add(basePort + i);
30 }
31
32 // Criando uma nova transforma o
33 try {
34     MiniTransformProxyN transf = new MiniTransformProxyN(
35         gatIp, 5500, "transf", "teste", nInputs,
36         ips, ports, finalPort, 9070
37     );
38
39     Thread.sleep(1000);
40     transfUid = transf.getUid();
41 }
42 catch (Exception e) {
43     System.err.println("error on mainminiencoder.java");
44     System.exit(0);
45 }
46
47 // Criando um controlador
48 VideolibEndpoint vle = new VideolibEndpoint(gatIp, 5500, "teste");
49
50 try {
51     // Esperando o controlador popular sua lista de componentes
52     System.out.println("Sleeping 5s");
53     Thread.sleep(5000);
54 } catch (InterruptedException e) {
55     e.printStackTrace();
56 }
57
58 // Coleta lista de v deos fonte
59 List<VideolibSource> srcs = vle.getSources();
60
61 // Conectando v deos transforma o criada anteriormente
62 for (int i = 0; i < nInputs; i++) {
63     vle.getTransform(transfUid).getVideoFrom(
64         srcs.get(i % srcs.size()).getUid(), i);
65 }
66 }
67 }
```
